Universidad Zaragoza
1542

Proyecto Fin de Carrera

Automatización del control, mantenimiento y localización de vehículos

Autor

Javier Torres Sebastián

Director y ponente

Ernesto Sánchez Pano

Antonio Romeo Tello

Escuela de Ingeniería y Arquitectura

2015

AUTOMATIZACIÓN DEL CONTROL, MANTENIMIENTO Y LOCALIZACIÓN DE VEHÍCULOS

RESUMEN

Este proyecto consiste en el diseño de un sistema que pueda extraer de un vehículo, de manera automática, la información necesaria para conocer su funcionamiento, con el objetivo de utilizarla para distintos fines.

El sistema consta de un cable especial que permite la conexión con un vehículo y de una placa de hardware libre proporcionada por la empresa "CUSTODIUM Soluciones y Sistemas S.L.", en la cual se ha realizado este proyecto.

El cable se enchufa al puerto OBD-II presente en cualquier vehículo y lee la información necesaria de la centralita, la cual pasa a través del cable a la placa de hardware libre, donde puede ser tratada.

Esta placa incluye distintos módulos y dispositivos que permiten grabar en ella un programa que se ejecutará indefinidamente realizando las funciones deseadas. Además, dos de estos módulos son, un sistema de comunicación que puede enviar datos a un servidor web y una antena GPS, los cuales proporcionan al sistema completo ventajas muy útiles.

El desarrollo de este proyecto consiste en el diseño del programa que irá grabado en la placa y los posteriores ensayos en vehículos. En cuanto al programa, realicé la parte del código que obtiene datos de la centralita, programé distintas alertas que avisaran de algún fallo utilizando esta información, incorporé dichos datos y alertas en un comando encargado de enviarlos (subirlos a internet) y diseñé alguna función extra para darle más utilidad al programa.

El código resultante fue añadido al programa diseñado por "CUSTODIUM Soluciones y Sistemas S.L." para su placa, el cual se encarga de la localización GPS, la optimización de la batería, el envío de varios datos y otras funciones. Juntos forman el programa utilizado para los ensayos en vehículos, durante los cuales se han realizado diversos cambios para mejorar el sistema.

La finalidad de este proyecto es obtener una herramienta que las grandes empresas puedan instalar en su flota de vehículos, con el objetivo de conocer mejor su funcionamiento para poder reducir costes, prevenir o identificar averías y localizarlos vía GPS si fuese necesario, pudiendo consultar toda esta información, en tiempo real, desde cualquier dispositivo con conexión a internet.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN

La incorporación de sistemas electrónicos en los vehículos hace necesario un sistema de autodiagnóstico capaz de detectar, evaluar y almacenar fallos. Cuando se empezó a desarrollar el sistema, este funcionaba en la forma específica del fabricante, pero con el aumento de la severidad de los gobiernos con la intención de reducir el peligro de los gases de escape, los legisladores vieron en el autodiagnóstico un medio auxiliar de supervisión de estos gases e introdujeron una estandarización independiente del fabricante. Este sistema, integrado actualmente en todos los vehículos, se denomina sistema OBD (On-Board Diagnostics System).

La unidad de control electrónico (ECU), también llamada centralita, contiene varias funciones de diagnóstico que identifican los fallos relevantes del sistema que puedan aumentar los gases de escape. En cada país se fijan diferentes valores límite para los gases contaminantes, que si se sobrepasan, encienden un testigo de averías para alertar al conductor. La lectura de los registros de averías se realiza mediante un enchufe de diagnóstico que permite la comunicación con la unidad de control mediante un protocolo de comunicación.

Existen hasta 100 funciones de diagnóstico distintas que hoy en día puede tener un vehículo, también llamadas PID's (Performance Information Data), que permiten obtener información de los diferentes parámetros, pero no todas ellas suelen estar disponibles para consulta. Lo cierto es que la ley únicamente obliga a los fabricantes a incorporar en los vehículos un sistema de autodiagnóstico y un enchufe OBD con un protocolo estándar, pero las funciones que contiene la ECU y los datos que se pueden consultar a través de dicho conector, son realmente decisión de cada fabricante. De hecho, además de las 100 funciones genéricas conocidas, cada marca tiene alguna función propia con su código correspondiente.

Aunque la finalidad original de estas funciones de diagnóstico fuese la de comunicar un aumento en los gases de escape para poder corregirlo, hoy en día suponen una herramienta mucho más útil, usada por los talleres de reparaciones para detectar todo tipo de averías. Existen en el mercado multitud de comprobadores que, conectados al enchufe OBD, ofrecen al técnico del taller un amplio abanico de posibilidades para encontrar la avería lo más rápidamente posible.

Pero quizá podamos ir más lejos. Si la información obtenida del vehículo, además de usarse para detección de errores como hacen estas máquinas de diagnóstico, fuese leída, almacenada y utilizada para otro fin, se obtendría una valiosa herramienta de gestión, que es lo que se ha diseñado en este proyecto. Podría crearse un programa que utilizando estos datos, avise por ejemplo cuando el vehículo supera una velocidad límite, cuando la temperatura del motor es muy elevada y es necesario detenerlo, cuantas horas seguidas lleva conduciendo el conductor, cuantas cargas de depósito se

han realizado o evidentemente, que alerte cuando exista una avería. Esto a priori no parece muy útil cuando uno va dentro del propio vehículo, ya que este dispone de paneles y testigos que avisan de ello, pero puede resultar muy ventajoso para el empresario que tiene que gestionar una gran flota desde su despacho. También podría ser de mucha utilidad para otros sectores con otras necesidades como se verá más adelante.

El proyecto que se expone a continuación consiste por tanto en la implementación de un software, que utilizando los datos extraídos de la centralita del vehículo en tiempo real, genere de manera automática determinadas alertas cuando sea necesario y envíe a un servidor de internet o a una aplicación móvil dichas alertas y otros datos interesantes, que otra persona puede consultar en cualquier parte del mundo.

Este proyecto ha sido realizado en la empresa "CUSTODIUM Soluciones y Sistemas S.L." utilizando sus materiales y herramientas y solicitando ayuda en alguna ocasión aislada cuando ha sido necesaria para realizar alguna tarea concreta.

2. OBJETIVOS

Los objetivos principales del proyecto, por medio de los cuales se ha conseguido el fin propuesto, son los explicados a continuación.

En primer lugar se debe diseñar un programa utilizando el software de la plataforma Arduino, escrito en el lenguaje C++, que comunique con la ECU del vehículo, lea los datos y los introduzca en varias funciones, para que devuelvan una alerta en caso de que algo no funcione como es debido. El programa también debe contener una función que envíe toda la información obtenida.

Este programa ha de introducirse en un microcontrolador contenido en alguna plataforma de hardware libre, que realice de manera autónoma las funciones requeridas. La empresa CUSTODIUM me ha proporcionado una placa de estas características diseñada por ellos, que además de las funciones de cualquier otra plataforma similar, tiene un sistema de localización GPS (Global Positioning System).

El siguiente objetivo es probarlo en el vehículo y corregir los errores que surjan, tanto de lectura como de posterior comunicación. Para ello habrá que probar con distintos instrumentos, distintos vehículos, diferentes posiciones de cables y pequeñas modificaciones de código, para conseguir finalmente, no solo que el sistema funcione, sino también que los resultados sean coherentes.

Por último, se realizará un seguimiento GPS del vehículo y múltiples ensayos en diferentes condiciones, para obtener datos en situaciones extremas que sean representativos y muestren perfectamente el alcance del proyecto.

Todo esto llevará a la consecución de un objetivo final y general que consiste en la creación de un sistema que las empresas puedan instalar en los vehículos de su flota, con la idea de controlarlos vía GPS y conocer el uso que sus empleados hacen de ellos, para poder reducir costes, prevenir riesgos, evitar fraudes y robos, y poder controlar a distancia desde una oficina otros aspectos como averías, excesos de velocidad o incluso períodos de ITV.

3. ANÁLISIS HISTÓRICO

En 1966, para combatir el problema de contaminación en la cuenca de Los Ángeles, el estado de California obligó a instalar sistemas de control de emisiones en los automóviles.

El gobierno federal extendió estos sistemas a nivel nacional en 1968, pero la primera empresa que los introdujo en la fabricación de sus vehículos fue Volkswagen en 1969, que incorporó un sistema con capacidad de escaneo en su modelo “Type 3”.

Figura 1. Volkswagen Type 3.

Figura 2. Volkswagen Type 3.

Un año después, en 1970, el congreso Norteamericano aprobó la ley de aire limpio y se estableció la Agencia de Protección Ambiental (EPA). Con esto empezó la publicación de distintas normas de emisiones que trataban de reducir la contaminación de los vehículos por medio de ciertas especificaciones.

En 1975, los ordenadores de a bordo (ajenos a los pasajeros) comenzaron a aparecer en los vehículos de consumo, como es el caso del “Datsun 280Z”, una de las variaciones del modelo “Nissan S30”, que fue uno de los primeros en llevarlo

incorporado. Esta práctica fue motivada en gran parte por la necesidad de un ajuste en tiempo real de ciertos sistemas del vehículo, lo cual era necesario para poder cumplir las especificaciones de la EPA. Dichos ordenadores consistían en sistemas electrónicos de alimentación y encendido de combustible, donde unos sensores medían el rendimiento del motor y ajustaban los mecanismos necesarios para reducir la contaminación. Estos mismos sensores son los que más tarde se usaron también para diagnosticar todo el vehículo. Aparecen también otras implementaciones simples, aunque no hay una estandarización en lo que se controla o cómo se muestra, sino que cada fabricante tenía sus propios sistemas y señales.

Figura 3. Nissan S30, Datsun 280Z.

Figura 4. Nissan S30, Datsun 280Z.

Posteriormente, en 1980, General Motors implementó una interfaz propia y un protocolo para realizar pruebas con la centralita (ECU), el cual incorporó en la línea de ensamblaje de sus vehículos. Este sistema detectaba de manera automática fallos en el motor y avisaba al usuario encendiendo una luz llamada "Malfunction Indicator Lamp" (MIL). Al poco tiempo se empezó a introducir en todos los vehículos, y hoy en día todavía se usa. Los códigos de diagnóstico (DTC), que distinguen una avería de otra, podían ser interpretados a través del patrón de parpadeo de esta luz.

Figura 5. Malfunction Indicator Lamp (MIL).

Varios años después, en 1988, La Sociedad de Ingenieros Automotrices (SAE) recomendó incorporar en los vehículos un conector para realizar diagnósticos externos

y un conjunto de señales necesarias para ello. Siguiendo su consejo, la California Air Resources Board (CARB) determinó que todos los vehículos de gasolina vendidos en California a partir del año 1991 debían tener, además del sistema automático de detección de averías y corrección de emisiones, dicho conector que permitía consultar información del automóvil y monitorear algunos componentes relacionados con la contaminación, sistema que recibió el nombre de OBD (On-Board Diagnostics). Pero el tipo de conector y su posición en el vehículo, así como el protocolo de datos, todavía no estaba estandarizado.

Motivados por el deseo de un programa estándar de pruebas de emisiones en todo el estado, la CARB publicó en 1994 una serie de especificaciones y mandatos que debían ser adoptados por todos los vehículos vendidos en California desde 1996. Esta publicación contenía una serie de códigos DTC sugeridos por la SAE y un conjunto ampliado de normas, las cuales eran más estrictas que las exigidas hasta entonces en lo referente a los límites de emisión.

Esto supuso la creación de un nuevo sistema, cuya finalidad era: estandarizar el conector y el protocolo de datos usado, corregir ciertas deficiencias del sistema anterior y hacer que fuera más fácil de usar para los técnicos de servicio y profesionales de talleres. Respecto al anterior, este sistema controlaba el motor casi completamente para reducir las emisiones y aumentar el rendimiento, activaba automáticamente distintos sistemas de seguridad (ABS, ESP, etc), tenía un registro de averías anteriores y monitoreaba a través del conector: partes del chasis, el cuerpo, los accesorios y la red de diagnóstico del coche. Se le puso el nombre de OBD-II para diferenciarlo del sistema anterior, que desde ese momento pasó a llamarse OBD-I. En 1996, este nuevo sistema fue obligatorio introducirlo en todos los vehículos fabricados, no solo en California como estaba previsto, sino en todos los Estados Unidos.

Figura 6. Puerto OBD-II.

Figura 7. Puerto OBD-II.

Poco después, en Europa, se introdujo un sistema muy similar ajustándose al OBD-II americano, al cual se le dio el nombre de EOBD (European On-Board Diagnostics) y la

Unión Europea obligó a todos los fabricantes a incorporarlo en los coches de gasolina fabricados desde el año 2001 y en los diésel desde el año 2003.

Con el paso de los años, la normativa se fue extendiendo a todo tipo de vehículos. En Norteamérica por ejemplo, fue obligatorio el sistema OBD-II para vehículos de tamaño medio desde el año 2005 y para vehículos pesados desde el año 2010, con el nombre de HDOBD (Heavy Duty On-Board Diagnostics). Incluso Japón sacó su propio sistema llamado JOBD, basado en los ya existentes.

4. HERRAMIENTAS Y COMPONENTES USADOS

4.1. CONECTOR OBD-II

La primera de las herramientas usadas y probablemente la más importante del proyecto es el conector OBD-II, actualmente presente en cualquier vehículo. Este conector, como ya se ha explicado anteriormente, es la puerta al exterior de un sistema integrado en la centralita, el cual analiza todos los parámetros, detecta fallos, memoriza averías, modifica automáticamente ciertos elementos para reducir emisiones y aumentar la seguridad, y por último, muestra la información a quien la requiera a través de este terminal.

El conector OBD-II se encuentra ubicado, desde su estandarización en 1996, debajo del volante, en la caja de fusibles o muy próximo a ellos. Dependiendo de la marca y el modelo del vehículo, puede encontrarse a simple vista u oculto por una carcasa que habrá que retirar para acceder a él.

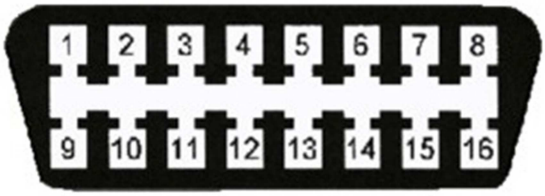
Actualmente existen tres protocolos básicos diferentes, mediante los cuales el sistema OBD-II se comunica con la centralita del vehículo y muestra la información a través del conector. Estos tres protocolos, utilizados en la mayoría de los coches, a su vez pueden tener pequeñas variaciones que dan lugar a otros muy similares, pero a pesar de esta variedad de códigos el sistema sigue siendo suficientemente estándar, ya que impide que cada fabricante introduzca en sus vehículos lenguajes nuevos y les obliga a elegir entre uno de los pocos existentes. Además, cada marca ha escogido un protocolo a utilizar y todos los vehículos que salen de su fábrica salen con el mismo, por tanto es fácil saber qué tipo de protocolo usa cada coche.

El primero de estos protocolos básicos, y uno de los más usados, es el protocolo ISO 9141-2, utilizado en la mayoría de los vehículos europeos, en todos los asiáticos y en los de la marca Chrysler, que además tiene una variante llamada ISO 14230, que es la utilizada por la marca Renault.

Los otros dos protocolos básicos pertenecen al tipo SAE, diseñado por la Sociedad de Ingenieros Automotrices, y fueron los primeros protocolos estándar que existieron para el sistema OBD-II. Estos son, el SAE J1850 VPW (Ancho de Pulso Variable), utilizado por GM USA (General Motors), y el SAE J1850 PWM (Modulación de Ancho de Pulso), utilizado por Ford USA.

También fueron saliendo posteriormente otras variantes muy parecidas, como el protocolo KWP, con sus variantes KWP 1281 y KWP 2000, utilizadas por el grupo VAG (Volkswagen AG), y los protocolos CAN bus.

Los conectores OBD-II están formados por 16 pines, a través de los cuales se conectan a un dispositivo de lectura y comunican la información al exterior. Debido a la estandarización de los conectores, estos pines se encuentran en todos ellos independientemente de la marca, el modelo o el protocolo usado, pero la función de algunos puede variar.



1 – Sin uso	9 – Sin uso
2 - J1850 Bus positivo	10 - J1850 Bus negativo
3 – Sin uso	11 – Sin uso
4 - Tierra del Vehículo	12 – Sin uso
5 – Tierra de la Señal	13 – Tierra de la señal
6 - CAN High	14 - CAN Low
7 - ISO 9141-2 - Línea K	15 - ISO 9141-2 - Línea L
8 – Sin uso	16 - Batería - positivo

Figura 8. Pines del puerto OBD-II.

Como se puede ver en esta imagen, cada pin tiene su función específica, pero en aquellos vehículos que por ejemplo utilicen el protocolo SAE J1850, el pin 7 y 15 no aparecerá en su conector o estará desenchufado, mientras que en aquellos que utilicen el protocolo ISO 9141, serán los pines 2 y 10 los que no aparecerán o no estarán conectados al sistema. Para el resto de protocolos menos comunes, las funciones de los pines serán prácticamente las mismas, a excepción de alguna pequeña variación de este estilo.

Como ya se ha dicho anteriormente, la ley solo obliga a la incorporación de un sistema de autodiagnóstico y un enchufe OBD-II con un protocolo estándar, pero

posteriormente cada fabricante puede ir más allá. El protocolo establece una serie de parámetros comunes que después cada marca puede ampliar con algunos propios. Estos parámetros son los que muestran la información del estado del vehículo proporcionada por múltiples sensores, pero no todos ellos se pueden consultar a través del enchufe. La lista completa de parámetros (PID's) se puede ver en el *Anexo 1*, pero los utilizados y mostrados por cada vehículo, son decisión del fabricante.

4.2. ADAPTADOR OBD-II

Desde la implantación del conector en los vehículos, se empezaron a diseñar distintos adaptadores que permitían realizar un diagnóstico externo de emisiones y averías y monitorizar los datos obtenidos. Estos adaptadores suponían una notable ventaja para los técnicos de reparaciones, que eran prácticamente los únicos que disponían de ellos y que podían aprovecharse de su utilidad.

Consistían en una sofisticada y muy cara máquina de diagnosis que mostraba en una pequeña pantalla los datos necesarios para la reparación.

Figura 9. Máquina de diagnosis.

Figura 10. Máquina de diagnosis.

Posteriormente, surgieron otros adaptadores que permitían mostrar en un ordenador los datos del vehículo, conectándose a este mediante puerto USB (Universal Serial Bus). Pero para poder leer y analizar dichos datos en el ordenador se necesitaba un programa específico, que también resultaba bastante caro y que solo los técnicos de reparaciones eran capaces de rentabilizar.

Con el paso de los años, debido a la demanda de la gente por querer conocer esta información, no solo con fines de reparación, empezaron a aparecer muchos adaptadores y programas de lectura más baratos y simples a nivel de usuario. Surgieron sencillos kits de diagnóstico con la máquina y el cable necesario, adaptadores con USB muy económicos que incluían el programa de ordenador e

incluso conectores con bluetooth que permitían la lectura de datos desde un teléfono móvil utilizando una aplicación disponible.

Figura 11. Kit de diagnosis OBD.

Figura 12. Cable OBD con USB.

Figura 13. Adaptador Bluetooth.

Pero el adaptador utilizado en este proyecto, aunque parecido, no es ninguno de los anteriores. Consiste en un cable compatible por un extremo con el conector OBD-II pero en el otro extremo dispone de cuatro pines. Este cable, llamado "OBD-II UART ADAPTER", tiene dos modelos diferentes: el modelo A, utilizado en este proyecto (por ser el único disponible en mi lugar de trabajo); y el modelo B, idéntico al A pero con un precio mayor y con un acelerómetro, un giróscopo y un sensor de temperatura, tres sistemas que no se necesitan para la realización del proyecto.

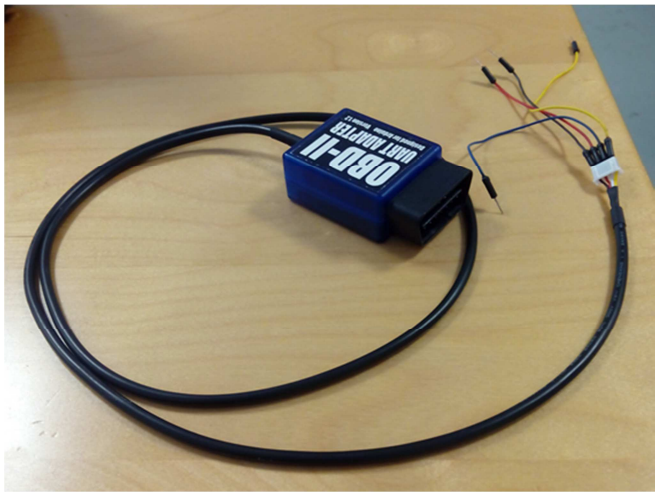


Figura 14. OBD UART Adapter, modelo A.

Como se ve en la imagen, los cables que contienen los cuatro pines son de colores diferentes. El cable rojo contiene el pin de alimentación, que alimentará el dispositivo al que se conecte utilizando la energía eléctrica de la batería del coche; el cable negro contiene el pin de tierra, para poder tener la misma referencia de tensiones en el vehículo y en el dispositivo conectado a él; y por último, los cables azul y amarillo

contienen los pines de comunicación, el amarillo (TX) manda datos y el azul (RX) los recibe.

Las ventajas de este adaptador, en comparación con los nombrados anteriormente, son numerosas. En primer lugar, la limitada longitud de algunos cables permite únicamente consultar datos dentro del vehículo. Incluso utilizando el adaptador con bluetooth, la distancia máxima a la que funciona es de 10 metros. Mientras que el cable utilizado para este proyecto permite la conexión a un dispositivo que, como se verá más adelante, puede enviar datos de manera automática a un servidor de internet o a una aplicación móvil, que permiten su consulta desde cualquier lugar del mundo. Además, el hecho de poder enviar los datos a internet permite poder recoger la información de varios vehículos simultaneamente, mientras que con otros adaptadores solo se puede extraer información del vehículo al que se conecta el cable o el sistema bluetooth. Pero la ventaja más importante, es la posibilidad de trabajar con la información obtenida del coche. Con los otros sistemas de lectura únicamente se puede visualizar la información en una pantalla o detectar alguna avería, pero con este cable, gracias a los pines vistos anteriormente, se puede pasar la información leída a una placa que se verá más adelante, capaz de realizar cálculos, generar alertas y enviar los resultados.

Respecto a los protocolos ya nombrados mediante los cuales se permite la comunicación con la centralita del vehículo, los compatibles con este cable son: CAN bus, ISO 9141-2 y KWP 2000.

4.3. ARDUINO

Otra herramienta clave para este proyecto, aunque no se encuentra presente en el resultado final, es la placa Arduino.

Arduino es una plataforma de electrónica abierta para la creación de prototipos basada en software y hardware flexibles y fáciles de usar. Se creó para artistas, diseñadores, aficionados y cualquiera interesado en crear entornos u objetos interactivos. A este tipo de dispositivos también se les conoce como plataformas de hardware libre.

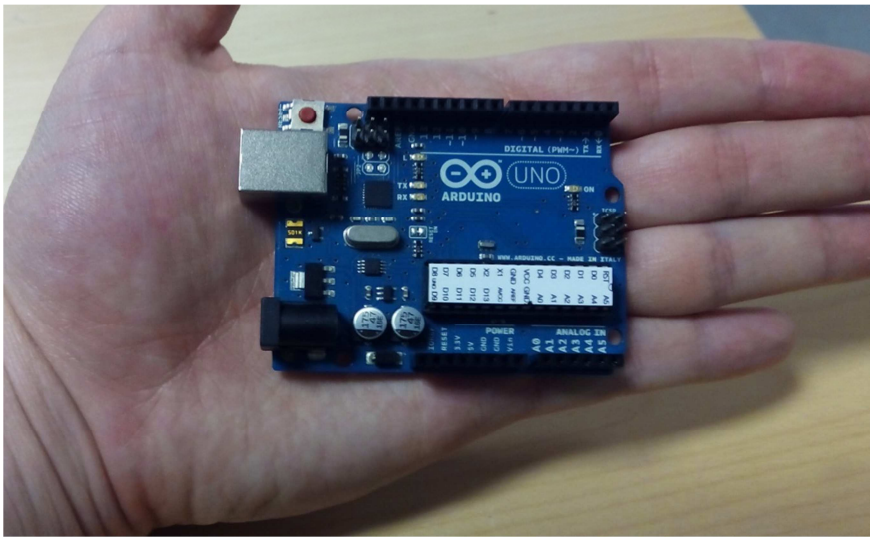


Figura 15. Placa Arduino UNO.

Arduino puede obtener información del entorno a través de sus pines de entrada, a los cuales se pueden conectar diversos sensores existentes en el mercado, pero también puede actuar sobre luces, motores y otros actuadores conectados a sus pines de salida. Para ello, estas placas de hardware libre disponen de un microcontrolador que se programa mediante el lenguaje Arduino, para realizar de manera automática y repetitiva aquello que se desea que haga.

Este lenguaje es básicamente el lenguaje C++, pero el software de Arduino, también llamado Arduino IDE (Integrated Development Environment), facilita el proceso de programación, detecta fallos e incoherencias, permite añadir librerías (que consisten en pequeños programas predefinidos) y finalmente graba el programa en el microcontrolador. También dispone de un monitor serie que muestra la información que recibe la placa por dicho puerto. Este software y las librerías necesarias pueden ser descargados de forma gratuita.

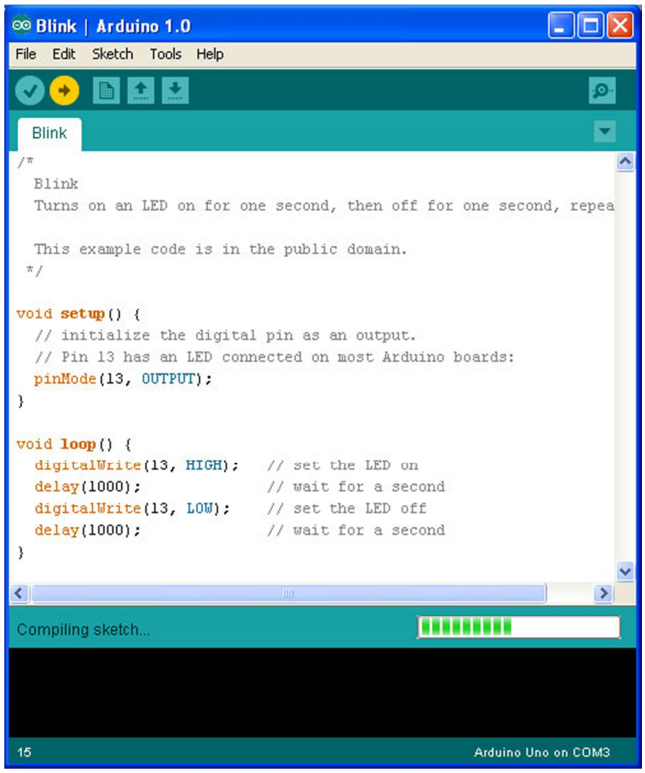


Figura 16. Arduino IDE.

El proceso de programación empieza con la escritura del programa en el software. Una vez finalizado y comprobada la ausencia de fallos, se conecta la placa Arduino al ordenador mediante un cable USB y se graba en su microcontrolador utilizando dicho software. Desde ese momento el programa se repetirá indefinidamente siempre que se alimente la placa de manera correcta, ya sea con el puerto USB del ordenador o con cualquier otra fuente de alimentación.

Desde su aparición en 2005, Arduino ha ganado importancia y ha desarrollado más de 20 tipos diferentes de placas, con características específicas para cada modelo pero todas con una misma función general. En este proyecto se ha utilizado el modelo Arduino UNO, ya que cubría las necesidades previstas, y al estar disponible en el lugar de trabajo, evitaba la adquisición de una nueva placa.

Arduino recibió una Mención Honorífica en la sección *Digital Communities* de la edición del 2006 del *Ars Electronica Prix*.

4.4. CUSTODIUM TRACKER

"Custodium Tracker" es el primer localizador GPS en miniatura con hardware libre, que dispone de toda la documentación, soporte y ejemplos necesarios para que el usuario final pueda adaptarlo a sus necesidades específicas, haciendo que se comunique con

otros dispositivos, sensores, actuadores, etc. Es un producto creado por la empresa "CUSTODIUM Soluciones y Sistemas S.L.", en la cual ha sido desarrollado este proyecto.

Figura 17. Custodium Tracker (Cara 1).

Figura 18. Custodium Tracker (Cara 2).

Además de los componentes frecuentes en cualquier plataforma de hardware libre, Custodium Tracker dispone de: un reloj en tiempo real con su propia batería, el sistema GPS ya citado, un módulo de comunicaciones que incluye un conector para antena y alojamiento para tarjeta SIM, y una conexión que permite la incorporación de una batería que alimente todos los elementos de la placa, otorgándole autonomía y evitando la necesidad de una fuente externa. Las características y componentes de esta placa se muestran con más detalle en el *Anexo 2*.

Gracias a su diseño modular, se le puede incorporar como ya se ha dicho: una batería, antena GPS, antena GSM (Global System for Mobile communications) y también diferentes módulos, tales como: un sistema Wireless (Wifi, bluetooth, Xbee), una placa de expansión (que incluye más puertos y conexiones) y otros muchos en desarrollo.

Respecto al software, en lugar de crear una plataforma nueva, esta empresa ha querido mantener la compatibilidad de Custodium Tracker con el IDE y ejemplos de Arduino, lo cual permite que cualquier desarrollador que ya tenga conocimientos de esta plataforma pueda usarlos directamente, además de poder consultar y recibir ayuda de la comunidad ya existente. Por tanto, el método para programar y grabar en la placa Custodium es idéntico al utilizado con Arduino.

El usuario o desarrollador tiene total libertad para elegir el proveedor de datos GPRS, la aplicación de internet o los mapas, pero existe un servidor web y una aplicación móvil creadas por la empresa, que se entregan junto con la placa y complementan las funciones de esta plataforma. Dichas aplicaciones han sido utilizadas para la realización de este proyecto.

Este servidor web, al que Custodium Tracker envía los datos, dispone de una aplicación de mapas en la que el usuario final puede ver la posición de varios trackers, las rutas que han seguido y otros datos interesantes, que se van almacenando en el servidor para posteriores consultas.

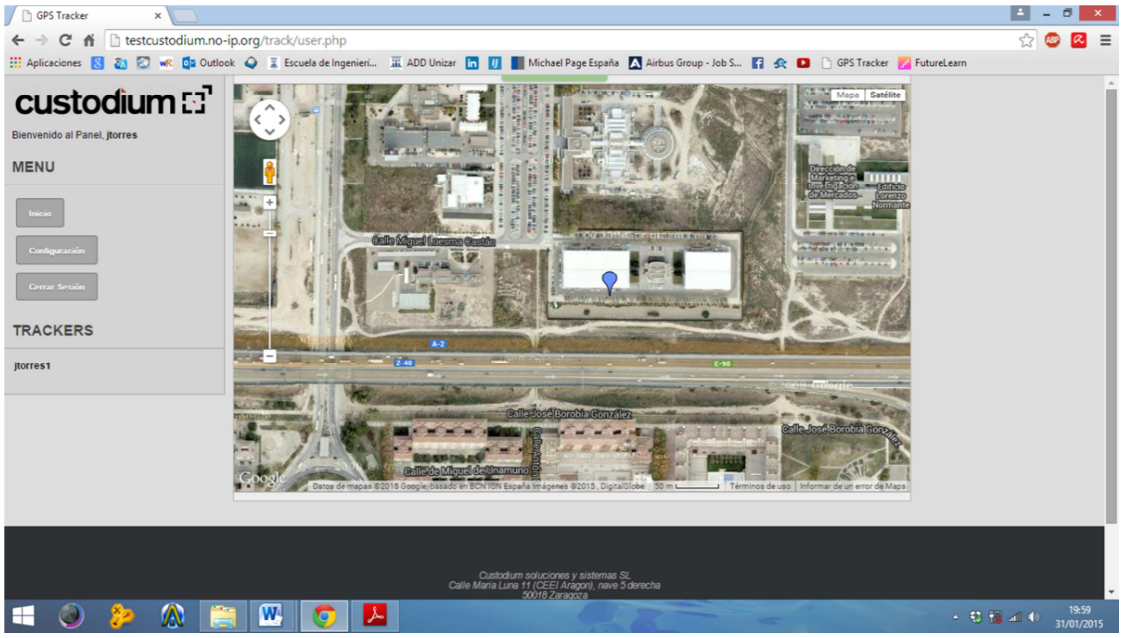


Figura 19. Servidor web de Custodium.

4.5. FTDI

La última herramienta de todas es el FTDI, cuyo nombre son las siglas de una empresa escocesa llamada Future Technology Devices International, especializada en tecnología USB.

Desarrollan y fabrican dispositivos semiconductores y sus correspondientes controladores de software para la conversión de transmisiones serie a señales USB, con el fin de permitir la compatibilidad de algunos dispositivos con los ordenadores modernos. Por tanto, su única contribución en el proyecto es la de permitir la conexión a un ordenador de aquellos elementos que no tengan el puerto necesario.

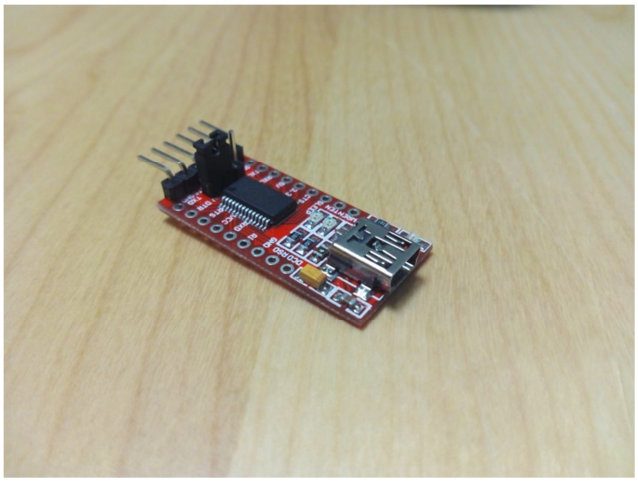


Figura 20. FTDI.

El FTDI realmente utiliza un conector micro USB, pero su conexión con el USB de un ordenador se puede realizar mediante un cable muy económico disponible en cualquier establecimiento de electrónica.

5. PROCESO

Como ya se ha comentado con anterioridad, la finalidad del proyecto es el diseño de un sistema electrónico y automático que obtenga información de un vehículo para tratarla y utilizarla, con el fin de obtener una sofisticada herramienta de gestión y localización cuyos datos pueden ser consultados desde cualquier dispositivo conectado a internet.

Para tener una idea inicial y aproximada de la tarea a realizar, existen diversos vídeos en internet, disponibles en la bibliografía, que muestran la obtención y visualización de los datos de un vehículo utilizando herramientas similares a las usadas aquí, pero con posibilidades más limitadas y fines diferentes. Se podría decir de alguna manera que la idea existía previamente, pero usándola como punto de partida, se ha conseguido diseñar en este proyecto un sistema mucho más complejo, con una combinación de herramientas y dispositivos que no existía hasta el momento y con unas características increíblemente útiles y muy poco frecuentes en el mercado (ver *Anexo 4*).

5.1. FAMILIARIZACIÓN

Para la realización de dicho proyecto, lo primero fue familiarizarse con todos los instrumentos necesarios que se iban a utilizar. Para ello me documenté por medio de tutoriales y manuales, en algunos casos proporcionados por los compañeros de la empresa Custodium y otras veces encontrados en internet, en páginas de especialistas o en el sitio web del fabricante.

Gracias a ellos aprendí las características básicas del cable OBD-II, su modo de funcionamiento, la manera de conectar sus pines, los códigos de comunicación que utiliza y la compatibilidad de dicho cable con determinados modelos de coche, según la cual debería funcionar para todo vehículo fabricado en Estados Unidos desde el año 1996 y para cualquier vehículo fabricado en Europa a partir del año 2001 si es de gasolina y desde el año 2003 en el caso de los diésel.

También aprendí el funcionamiento de las placas de hardware libre, empezando por las placas Arduino. En este proyecto, de todos los modelos diferentes, se ha utilizado la placa "Arduino UNO", ya que era la única placa Arduino disponible en mi entorno de trabajo, y además, diversos ensayos y análisis demostraron posteriormente que sus características eran adecuadas y suficientes para la realización del trabajo previsto. Se puede ver más información sobre esta placa y sus componentes en el *Anexo 2*.

Una vez leídos los manuales, me descargué gratuitamente el software Arduino IDE, el cual además de permitir la escritura, tiene múltiples ejemplos para practicar, que consisten en programas predefinidos muy simples que permiten hacer pruebas y aprender mejor el funcionamiento de este software. Realicé varios de ellos únicamente conectando la placa al ordenador mediante un cable USB, grabando los datos y alimentándola a través de ese mismo cable. Uno de estos ejemplos era un sencillo programa que hacía parpadear un LED con una frecuencia de un segundo.

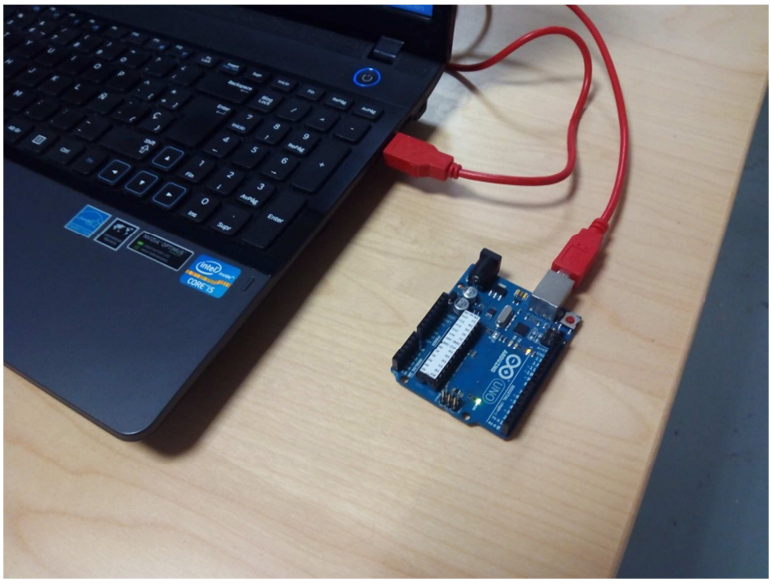


Figura 21. Conexión Arduino – Ordenador.

Posteriormente pasé a la utilización de la placa Custodium Tracker. Para aprender a usar dicha placa, además de leer los manuales, necesité alguna explicación de los compañeros de la empresa, ya que al ser un producto suyo, ellos me podían orientar y ayudar mucho mejor. También me proporcionaron varias antenas GPS, una antena GSM, una batería para la placa y me permitieron el acceso al servidor web y a la aplicación móvil, donde se envían todos los datos obtenidos.

Una vez familiarizado con el sistema, realicé mi primera prueba. Ya que el funcionamiento básico de esta placa es el mismo que el de otras plataformas de hardware libre, me interesaba probar principalmente las diferencias que esta tiene con todas las demás, es decir, su localización GPS y su módulo de comunicaciones. Para ello la empresa me proporcionó el programa que habían desarrollado como software implícito del producto. Este programa contenía un código muy complejo pero compatible con la plataforma Arduino IDE, la cual utilicé para grabar dicho programa en la placa Custodium.

Por motivos de espacio, con el fin de fabricar una placa manejable y reducida, se han eliminado aquellos elementos prescindibles o que solo se usan una vez, como los utilizados para grabar el programa. Por tanto, para poder conectar Custodium Tracker con un ordenador y cargar el código, es necesario utilizar un FTDI siguiendo el siguiente esquema.

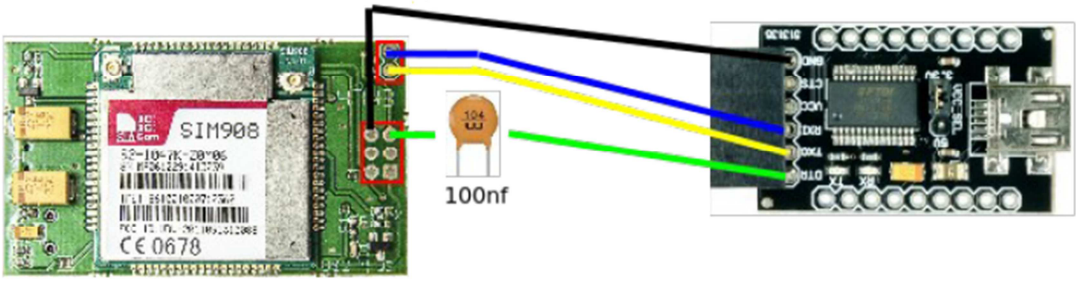


Figura 22. Conexiones FTDI - Custodium Tracker.

El cable negro conecta las tierras, para poder tener la misma referencia de tensiones en ambos dispositivos; el cable verde, que necesita un condensador, conecta ambos "reset"; y los cables amarillo y azul conectan los pines de comunicación TX y RX respectivamente. En este caso la placa Custodium no se debe alimentar, ya que lo hace la batería incorporada, pero de no ser así, debería conectarse un cable más.

Una vez grabado el programa, el siguiente paso era realizar una prueba, y para ello conecté la antena GPS y la antena GSM. Durante los ensayos con vehículo no es necesario el ordenador, pero resulta muy útil utilizarlo después de realizar algún cambio significativo en el código, como ocurre en este caso, ya que su uso permite visualizar el proceso con el monitor serie de Arduino IDE y se puede ver todo lo que hace el programa durante su ejecución, lo cual ayuda a detectar fallos que no se hayan tenido en cuenta.

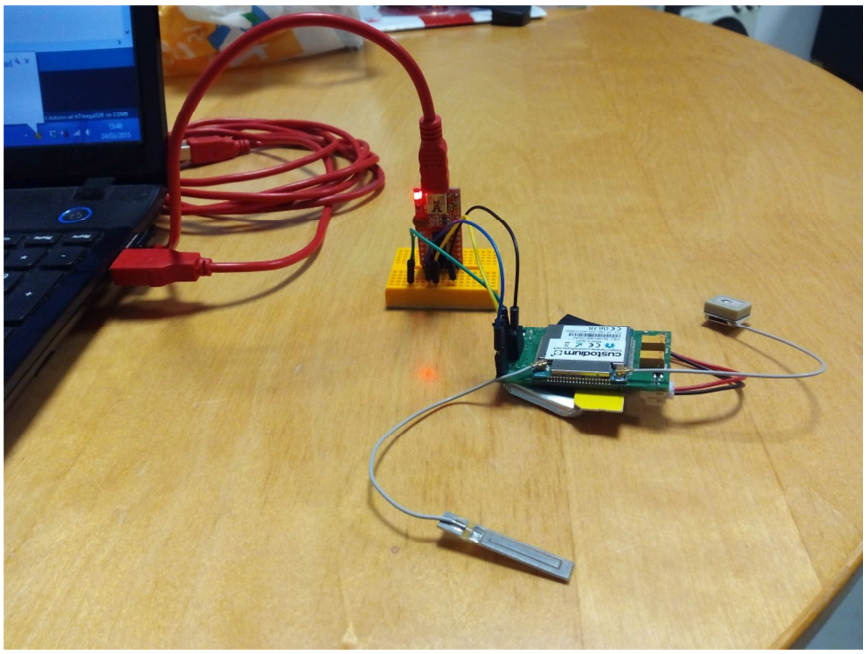


Figura 23. Conexión Custodium Tracker – Ordenador.

Con todo el sistema montado, salí al exterior para probarlo, ya que dentro de un edificio la señal GPS se atenúa o no llega, dando lugar a errores de localización. El ensayo fue exitoso como se preveía, ya que la empresa Custodium tenía muy perfeccionado su producto. Los datos de localización, fecha y estado de la batería se enviaron correctamente al servidor web, donde pude consultarlos fácilmente.

5.2. PROGRAMACIÓN Y ENSAYOS

Una vez conocidos los instrumentos del proyecto, el primer objetivo era diseñar un programa en lenguaje C++ utilizando Arduino IDE, que permitiese la comunicación con la centralita del vehículo a través del cable OBD-II. Para ello utilicé la placa Arduino UNO en primer lugar y posteriormente la placa Custodium Tracker, aunque en el proyecto final, únicamente aparece la segunda.

La razón de usar Arduino inicialmente es su simplicidad y sus múltiples posibilidades de conexión, ya que debido a la escasez de pines que tiene Custodium Tracker resulta imposible conectar el cable OBD-II y el ordenador al mismo tiempo. Esto impide utilizar el monitor serie de Arduino IDE nombrado anteriormente para observar la ejecución del programa paso a paso y ver la información leída por la placa. También existe el riesgo de no tener suficiente memoria RAM en Custodium Tracker para los programas de lectura de datos del coche, localización GPS y envío de la información al servidor.

Estos problemas debían ser solucionados posteriormente, pero en primer lugar, se utilizó la placa Arduino UNO a modo de prueba para comprobar el correcto funcionamiento del cable OBD-II conectado al vehículo, así como la posibilidad de tratar los datos obtenidos y comprobar de esta manera la viabilidad de la propuesta principal del proyecto. Además, dado que ambas placas utilizan el mismo lenguaje y el mismo software de programación, no habrá ningún problema al copiar el programa de una a otra posteriormente.

Lo primero que tuve que hacer antes de comenzar a programar fue descargar las librerías necesarias. Estas librerías consisten en programas predefinidos con características muy concretas, orientados a distintos dispositivos incorporados a la placa o a la realización de tareas adicionales del software. Se pueden entender como instrucciones complementarias, que deben ser llamadas desde el programa principal, para que este pueda realizar correctamente unas funciones determinadas. Pueden encontrarse en internet librerías de todo tipo en función de lo que se quiera realizar y su descarga generalmente es gratuita dependiendo del sitio web. Para este proyecto serán necesarias las librerías OBD.h y OBD.cpp, que añaden ciertos comandos al programa relacionados con el funcionamiento del cable, para que Arduino se pueda comunicar sin problemas con la ECU del vehículo. Estas librerías deben ser copiadas en la carpeta en la que se guarde el archivo del programa. Debido a la complejidad y al

gran tamaño del código que contienen, no se han introducido en el proyecto, pero en la bibliografía se puede encontrar la dirección web de la que han sido descargadas, para consultarlas si fuera necesario.

El siguiente paso fue realizar un programa muy sencillo para ensayar por primera vez en un vehículo. Este programa consistía en una lectura de las revoluciones del coche, de manera que cuando estas superasen la cifra de 2000, se encendiese un LED que la placa Arduino UNO tiene integrado en el pin 13. Para ello utilicé el PID "0x0C" de la lista del *Anexo 1*, pero para una mejor comprensión, la librería OBD.h asigna un nombre a los códigos más comunes, de manera que puedo utilizar las revoluciones en mi programa escribiendo "PID_RPM". Este programa se puede ver en el *Anexo 3.1*, con comentarios que explican la función de cada comando.

Una vez escrito el programa y grabado en la placa desde el ordenador, realicé mi primer ensayo. Para ello, un compañero de la empresa se ofreció a prestarme su vehículo, una furgoneta FIAT Dobló de primera generación.

Figura 24. Fiat Dobló.

En este primer caso no se necesitaba el ordenador para el ensayo, ya que bastaba con ver si el LED se iluminaba. Así que conecté el cable con la placa, colocando correctamente los pines de alimentación, masa y comunicación, y enchufé el adaptador al puerto OBD-II de la furgoneta. Teniendo todo conectado, encendí el motor y pude ver como se encendían dos luces en el cable: una roja permanente y una azul parpadeante. La roja indica la correcta alimentación gracias a la batería del vehículo, mientras que la azul indica que se están transmitiendo datos a través del cable.

Figura 25. Cable enchufado al vehículo. Luz roja de alimentación.

Respecto al programa realizado, en un principio el LED de Arduino UNO se encontraba apagado, ya que el ralentí de la furgoneta es menor de 1000rpm, pero al acelerar suavemente en punto muerto se podía observar que el LED se encendía cuando se superaban las 2000rpm y se volvía a apagar si dejaba de pisarse el pedal. Esta fue por tanto la primera prueba exitosa, que demostró el correcto funcionamiento de los componentes y la viabilidad del proyecto en general.

A continuación, para aumentar la complejidad, me dispuse a escribir otro programa que leyera mayor cantidad de datos, pero en este caso sí que se necesitaría un ordenador para visualizar los resultados. El problema era que el puerto utilizado para conectar Arduino UNO al ordenador es un puerto serie USB, que internamente en el hardware de la placa coincide con el puerto serie de los pines 0 y 1 (ver *Anexo 2*), a los que es necesario conectar los terminales de comunicación (amarillo y azul) del cable OBD-II, tal y como se había hecho en el ejemplo anterior. Esto supondría conectar ambos dispositivos al mismo puerto de Arduino UNO, de los cuales solo uno de ellos podría enviar o recibir información.

Por tanto, para poder conectar a la vez el ordenador y el cable, y dado que esta placa dispone de múltiples pines sin usar, se debe crear un puerto serie virtual en otros dos de ellos. La finalidad de esto es darle la función de puerto serie a dos pines que previamente no la tenían. Para ello se utiliza un comando llamado “SoftwareSerial”, para el cual se debe incluir una nueva librería en el programa (Arduino IDE ya dispone de ella, no es necesario descargarla) y adjudicarle a esta función dos pines que no sean el 0 y el 1. Elegí para ello, por ejemplo, lo pines 3 y 4. De este modo, el cable se conecta a los pines 0 y 1 como en el ejemplo anterior, ya que sus librerías lo exigen

(OBD.h y OBD.cpp), y en los pines 3 y 4 se conecta un FTDI que permite la conexión USB que necesita el ordenador.

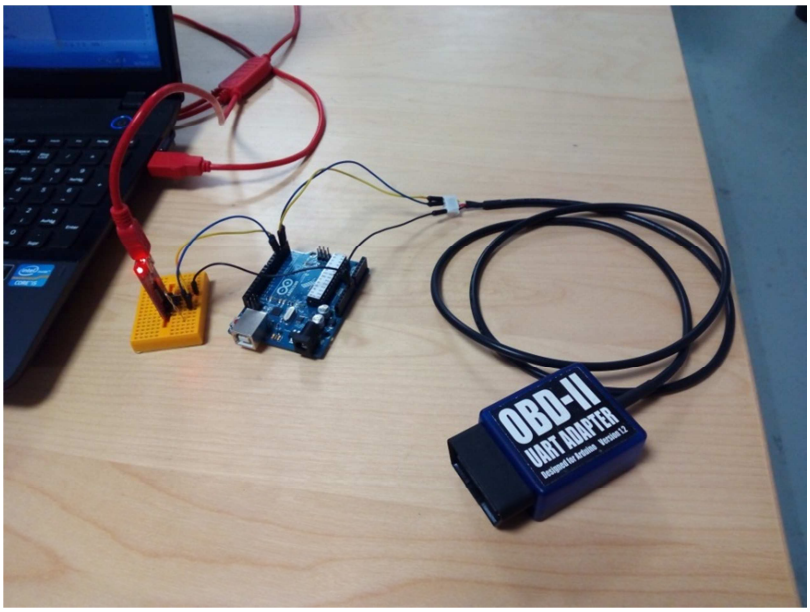


Figura 26. Conexión de Arduino con ordenador y cable OBD al mismo tiempo.

Este montaje resulta algo complejo, pero solo se utiliza provisionalmente para poder visualizar los datos leídos.

Una vez solucionado esto, comencé a escribir el programa, que en este caso debía contener comandos para la lectura de varios parámetros del vehículo y su escritura por puerto serie virtual. Para ello, seleccioné los códigos (PID's) que me resultaron más importantes de la lista completa (ver *Anexo 1*) y los metí en el programa con sus comandos correspondientes de lectura y escritura; introduje también las librerías OBD.h, OBD.cpp y SoftwareSerial; y adjudiqué los pines 3 y 4 al puerto serie virtual. Este programa se puede ver en el *Anexo 3.2*.

Con el programa finalizado y después de comprobar con Arduino IDE que no existían fallos, grabé todo en la placa y conecté el sistema completo en el vehículo, pero este ensayo no tuvo tanto éxito. El programa funcionaba y los parámetros se escribían correctamente en el monitor serie de Arduino IDE usado en el ordenador, pero la mayoría de ellos mostraban un valor igual a cero. Además, los valores que iban apareciendo se escribían a continuación de los anteriores, así que tras varios segundos era muy difícil distinguir los datos nuevos de los viejos, ya que la pantalla se saturaba de información.

El segundo problema lo solucioné fácilmente. En el monitor serie de Arduino IDE no se puede modificar nada, solo se puede visualizar la información que se recibe, pero existen otros programas en los que sí que se puede. Uno de ellos es el llamado Putty,

el cual pude descargar de forma gratuita. Tras buscar en diversas páginas y foros, conseguí encontrar un comando que borraba datos antiguos en Putty cuando recibía nuevos. En otras palabras, actualizaba los datos continuamente dejando la información mucho más clara en la pantalla.

El primero de los problemas resultó más difícil de resolver. Las causas de que los resultados tuviesen como valor un cero podían ser múltiples. Dado que algunos de los parámetros, como la velocidad y las revoluciones, funcionaban correctamente, descarté un error en el programa o en el cable. Otra causa probable podía ser que el modelo de la furgoneta fuese demasiado antiguo. Investigué las características de la Fiat Dobló y observé que este modelo fue fabricado por primera vez en Europa en el año 2000, aunque tuvo posteriores modificaciones en 2005. Según la normativa ya nombrada, el cable tendría que funcionar para todos los vehículos de gasolina fabricados en Europa a partir del año 2001, así que esta podría ser la causa. Era posible también, que se debiese a la elección por parte del fabricante de los comandos mostrados o al tipo de cable usado para leerlos. Para salir de dudas probé el sistema en otros dos vehículos.

Estos vehículos fueron un Nissan Almera Hatchback (modelo N16) y un Seat Ibiza de cuarta generación (modelo 6J).

Figura 27. Nissan Almera Hatchback.

Figura 28. Seat Ibiza.

Al realizar un ensayo en el Nissan, pude observar que ninguno de los parámetros mostraba un valor distinto de cero, ni siquiera aquellos que anteriormente habían funcionado. Este vehículo se fabricó por primera vez en Japón en el año 2000 y el protocolo más utilizado en ese país es el JOBD, que como ya se ha visto en el apartado “herramientas”, no es compatible con el cable OBD-II usado. Esta podría ser la causa de los valores nulos obtenidos, y como no era posible solucionar este problema, descarté este vehículo y decidí probar en el siguiente.

El Seat Ibiza de cuarta generación fue fabricado en Europa en el año 2008. Es suficientemente nuevo como para haberse diseñado según la normativa OBD, y el protocolo utilizado por este modelo concreto es el ISO 9141-2, que es uno de los reconocidos por el cable. De modo que en este coche debía funcionar mi sistema, pero no fue así. Afortunadamente, funcionaba mejor que en los dos vehículos anteriores, ya que un mayor número de los parámetros leídos mostraba un valor coherente, pero el resto seguían teniendo un valor nulo.

Basándome en el resultado de los ensayos realizados y tras mucho investigar en internet, llegué a la conclusión de que esto se debía a la decisión del fabricante, ya que es él quien decide qué parámetros, de la lista que aparece en el *Anexo 1*, deben mostrarse a través del conector. Pero existía una mínima posibilidad de que la causa fuese el modelo de cable usado. En cualquiera de los casos, no disponía de otras herramientas para realizar mi trabajo, así que eligiendo la mejor opción, decidí que el Seat Ibiza sería el coche en el que realizaría todos los ensayos y el proyecto en general.

Figura 29. Visualización de los datos del coche en el ordenador usando la placa Arduino.

Por tanto, el siguiente paso era comprobar todos los parámetros de esa lista, con el objetivo de ver cuáles de ellos están disponibles en este vehículo para su lectura.

Para ello realicé múltiples ensayos, introduciendo en mi programa todos los PID's de 10 en 10. Cada vez que realizaba una prueba, anotaba los parámetros que mostraban un valor coherente y distinto de cero, hasta obtener finalmente un total de 12 códigos disponibles para consulta.

En esa lista existen también parámetros de detección de averías, cuyo valor también era cero, pero no había manera de saber si ese resultado era correcto o no, ya que el coche utilizado no tenía ninguna avería. Como no había posibilidad de probarlo en algún vehículo con problemas, no resultaba muy útil usar dichas funciones.

Realicé por tanto un programa que incluía todos los parámetros válidos y adjunté en algunos de ellos diferentes alertas o comentarios, para que escribiesen por pantalla un mensaje en el caso de que alguno sobrepasara un valor preestablecido.

Para conocer mejor estos parámetros y poder determinar los valores límite para su correcto funcionamiento, investigué sus características en internet.

El primero de ellos muestra las revoluciones por minuto del motor, cuya utilidad ya se conoce y para las cuales establecí un límite subjetivo de 4000rpm. Considero que sobrepasar este límite durante un tiempo prolongado puede incrementar el consumo de gasolina y calentar demasiado el motor, lo que supone un riesgo para el vehículo. Por tanto, se muestra un aviso cuando se supera este valor.

El segundo parámetro es la velocidad del vehículo, cuyo funcionamiento también se sobrentiende y su valor límite es de 120km/h según el código de circulación. El programa muestra otra alerta cuando se excede dicho límite.

El siguiente es la temperatura del líquido refrigerante. Este líquido es el encargado de absorber temperatura del motor para mantenerlo entre 85ºC y 95ºC con el fin de que no se sobrecaliente. Para ello, una bomba se encarga de circular continuamente el líquido por los conductos internos del motor. Por tanto, la temperatura de este líquido realmente indica la temperatura del motor. Cuando supera los 95ºC ya nombrados, puede significar que el vehículo ha estado en funcionamiento un largo período de tiempo en condiciones extremas o que existe un fallo en el sistema de refrigeración. Sea cual sea la causa, puede suponer un riesgo y es necesario detener el vehículo. Tampoco es recomendable forzar el motor cuando su temperatura todavía es baja, ya que pueden producirse cambios bruscos de temperatura perjudiciales para ciertos elementos. Como conclusión, decidí establecer un mínimo de 80ºC, por debajo del cual se muestra un aviso, y un máximo de 95ºC, por encima del cual ocurre lo mismo.

Otro parámetro que incluí en el programa es el “valor calculado de carga”, que indica el porcentaje de par disponible para ser usado. El vehículo lo calcula en función de: la temperatura ambiente, el flujo de aire, la presión atmosférica y las revoluciones del motor. Como este parámetro es meramente informativo y no tiene límites de riesgo, no introduje ninguna alerta para él.

A continuación están los sensores de oxígeno, que sirven para poder conocer la calidad de la mezcla de combustible. Estos sensores leen la cantidad de oxígeno en el escape y le indican de manera automática al inyector de combustible (en motores diésel) o al

carburador (en motores de gasolina) cuánto combustible necesita el motor para mantener la proporción adecuada de aire/combustible, que debe ser de 14.7:1. El sensor 1 se encuentra antes del catalizador, mientras que el sensor 2 está situado después. Un mal funcionamiento de estos sensores puede causar: una degradación de este catalizador, una alta contaminación, problemas para arrancar el motor y un exceso en el consumo de combustible. Para evitar esto, se deben cambiar los sensores si se detecta alguna anomalía en los resultados. El valor que proporcionan dichos sensores está expresado en voltaje y debe oscilar entre 0 y 1V, pero para una mejor comprensión, los valores que muestra el cable están expresados en porcentaje, por tanto los datos oscilan entre 0 y 100%.

Este parámetro no tiene márgenes de peligro, pero si los resultados se mantienen estables en un valor alto o bajo y no oscilan al pisar el acelerador, puede deberse a una avería en los sensores. Ya que no se pueden establecer unos límites, tampoco se puede programar una alarma para detectar fallos y la única solución es observar cada cierto tiempo si los datos varían.

Otro parámetro medido es el llamado “fuel trim”, que básicamente es el porcentaje de cambio en el aporte de combustible a lo largo del tiempo. Este parámetro está por tanto estrechamente relacionado con las lecturas de los sensores de oxígeno, que se encargan de indicarle al motor si necesita mayor o menor cantidad de combustible para su mezcla. Existen dos valores diferentes en función del margen de tiempo: el primero es el llamado “Short Term Fuel Trim” (STFT), que indica el porcentaje de cambio instantáneo, dando varios resultados por segundo; y el otro se conoce como “Long Term Fuel Trim” (LTFT), que también muestra el porcentaje de cambio de combustible pero calculado en un período de tiempo mayor. En ambos casos, un porcentaje negativo indica una reducción del combustible aportado a la mezcla, que coincidirá con un voltaje alto de los sensores de oxígeno, mientras que un porcentaje positivo muestra un aumento de la cantidad de combustible añadida, que coincidirá por tanto con un voltaje bajo de los sensores.

Este parámetro es únicamente informativo y no puede avisar de ningún fallo en el vehículo, así que en el programa solamente he introducido un mensaje que indica si la mezcla es pobre en combustible, lo que significa que el motor está siendo forzado, por ejemplo por motivo de una pendiente; o si la mezcla es rica en combustible, lo que implica un funcionamiento relajado del motor.

El siguiente parámetro leído es la presión en el colector de admisión de aire, medido en KPa por el sensor MAP (Manifold Absolute Pressure), que permite determinar el flujo de aire entrante en el motor, un dato que el vehículo necesita para calcular el valor de carga disponible, como ya se ha explicado anteriormente. También guarda relación con la lectura de los sensores de oxígeno, ya que a mayor flujo de aire mayor cantidad de oxígeno será proporcionada a la mezcla del motor.

Otro de los parámetros es la temperatura del aire de admisión, medida en grados centígrados gracias al sensor IAT (Intake Air Temperature). Este sensor es utilizado por el vehículo para conocer la temperatura ambiente en un arranque en frío, con el fin de realizar automáticamente ciertos ajustes que permitan arrancar el motor sin problemas. También es necesario para que la centralita pueda calcular el valor de carga disponible y saber qué cambios debe realizar para seguir generando combustiones correctas y no perder rendimiento en el motor. Además, es un dato muy útil para los vehículos de refrigeración por aire, ya que les permite saber si la temperatura del aire entrante es suficiente para refrigerar el motor o si puede existir un sobrecalentamiento.

El siguiente parámetro leído es el avance de sincronización de chispa, expresado en grados de ángulo. La sincronización del encendido en un motor de combustión interna por chispa es el proceso de establecer el ángulo relativo de la posición del cigüeñal para la cual debe producirse la chispa en la cámara de combustión, según la velocidad angular en ese momento. La necesidad de adelantar o retrasar el momento de la chispa se debe a que el combustible no se quema por completo en el instante en el que esta se produce, los gases de combustión tardan un período de tiempo en expandirse, y la velocidad de rotación del motor puede alargar o acortar el período de tiempo en el que la quema y la expansión tienen lugar.

Este parámetro será descrito como un cierto ángulo antes del punto muerto superior (BTDC, Before Top Dead Center), punto donde la cámara de combustión llega a su tamaño mínimo y gracias a la detonación generada por la chispa, se expande bruscamente. Las chispas que se producen después del punto muerto superior (ATDC, After Top Dead Center) suelen ser contraproducentes, a no ser que exista la necesidad de una chispa suplementaria. Un correcto ajuste del tiempo de encendido es crucial para el rendimiento de un motor, ya que afecta a diferentes variables como el consumo de combustible y la potencia. Las chispas que se producen demasiado pronto o demasiado tarde en el ciclo del motor son a menudo responsables de vibraciones excesivas e incluso daños graves.

Para tener una idea de los valores de este parámetro cuando el sistema funciona correctamente, los datos leídos deben encontrarse aproximadamente en los siguientes márgenes:

Vehículo en movimiento sin acelerar ni frenar (Coasting): 0 grados
Punto muerto (Idle): 8-15 grados.
Crucero (<10% del acelerador): 30 a 40 grados
Aceleración leve (10-20% del acelerador): 30 grados o menos.
Aceleración moderada (20-40% del acelerador): aproximadamente 20 grados, pero puede caer a 18 momentáneamente.
Gran aceleración (40-xx% del acelerador): igual que el caso anterior.

Pedal a fondo (WOT, Wide Open Throttle): aproximadamente 15-25 grados, dependiendo de las RPM.

El último de los parámetros medidos en mi programa es la posición del acelerador, que no debe confundirse con la posición del pedal. Actualmente, en la mayoría de los coches, la posición del pedal únicamente controla un potenciómetro, el cual le dice a la ECU del vehículo la cantidad de presión que el conductor está realizando, pero es la centralita la que controla el acelerador después de realizar los cálculos necesarios. Por tanto se puede decir que no existe una relación directa entre ellos, y esta es la causa de que este valor, expresado en porcentaje, nunca llegue a 0%, ya que el sistema considera que para mantener el motor en funcionamiento debe activarse levemente el acelerador y consumir combustible. Por ello, a pesar de no pisar el pedal, el valor leído de este parámetro nunca descenderá de 10%.

Para estos cuatro últimos parámetros (presión en el colector de admisión de aire, temperatura del aire de admisión, avance de sincronización de chispa y posición del acelerador) no he introducido ninguna alerta ni comentario explicativo, ya que no se puede sacar ninguna conclusión ni detectar ninguna avería a partir de los resultados.

Como se puede observar, casi todos estos parámetros que el Seat Ibiza permite leer están muy relacionados unos con otros. La mayoría de ellos guardan relación con el rendimiento del motor, la potencia disponible o el consumo de combustible, datos que pueden resultar muy útiles para aquellos que quieran modificar su vehículo y que posean los conocimientos y herramientas para poder hacerlo. Esta es una de las principales razones por las que el sistema OBD-II y la lectura de datos han ganado fama en los últimos años. Pero para la idea principal de este proyecto, que consiste en informar a gran distancia de posibles averías, uso indebido o robo de vehículos, estos parámetros no parecen muy útiles.

A pesar de ello, a modo de prueba, realicé el programa ya nombrado con los comandos anteriores, incorporando sus alertas y comentarios correspondientes (Ver *Anexo 3.3*), y realicé diversos ensayos para comprobar que todo funcionaba correctamente, mostrando los resultados en el programa Putty.

Durante estas pruebas, el vehículo se encontraba detenido, con la palanca de cambio en punto muerto y los pedales sin pisar.

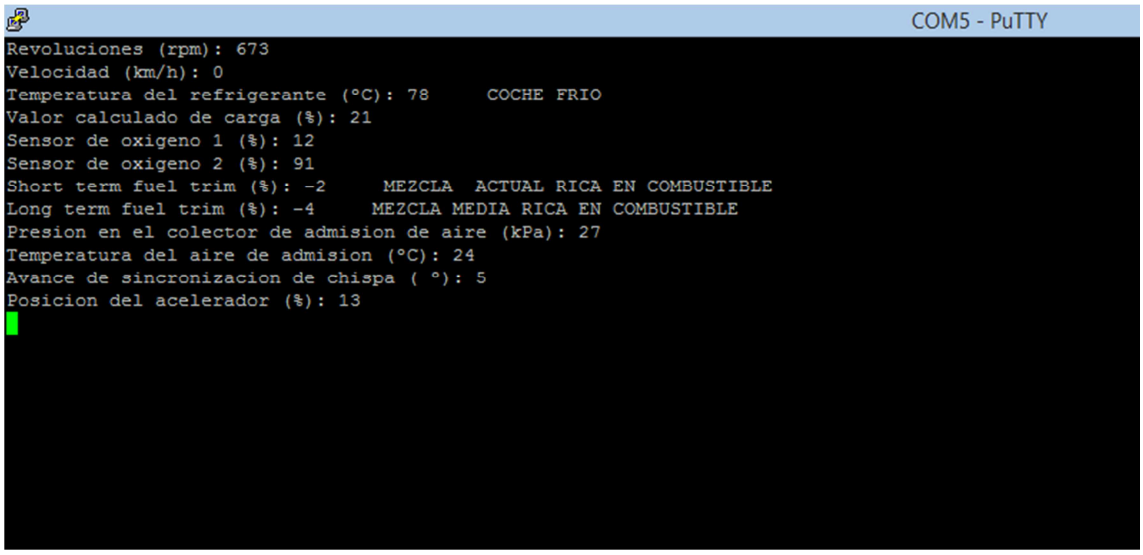


Figura 30. Datos extraídos del vehículo y alertas. Visualizados usando Putty.

Como se puede ver en esta imagen obtenida del primer ensayo (figura 30), debido al poco tiempo transcurrido con el motor en funcionamiento, la temperatura del refrigerante es inferior a los 80ºC establecidos como límite inferior, y por tanto, el programa escribe un aviso indicando que el coche está frío.

Se puede ver también que las revoluciones son bajas, ya que el coche se encuentra en régimen de ralentí; la velocidad es nula por estar detenido; el avance de sincronización de chispa es bajo por estar en punto muerto, aunque su valor es menor de lo esperado; y la posición del acelerador tiene un valor reducido pero sin bajar de 10%, tal y como se preveía.

Por último, los comandos de “short” y “long term fuel trim” son negativos porque el motor se encuentra en un estado relajado y el porcentaje de combustible que necesita el vehículo se reduce. Por ello, se puede ver que el programa muestra un mensaje indicando que la mezcla es rica en combustible.

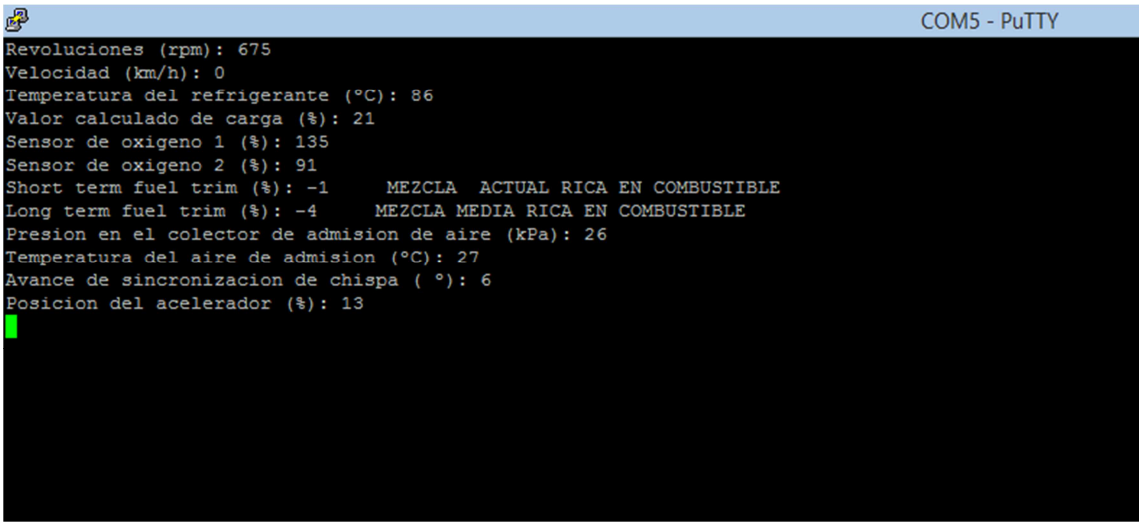


Figura 31. Datos extraídos del vehículo y alertas. Visualizados usando Putty.

En esta otra imagen (figura 31), los valores son prácticamente los mismos, pero al llevar más tiempo en funcionamiento el motor, su temperatura es mayor, y el aviso de "COCHE FRIO" ha desaparecido automáticamente por haberse superado los 80ºC.

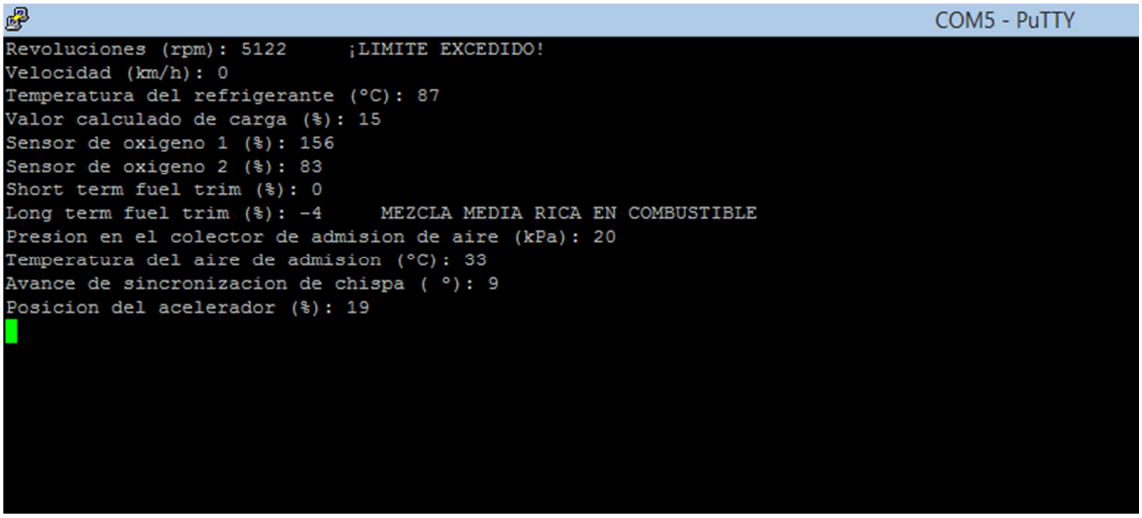


Figura 32. Datos extraídos del vehículo y alertas. Visualizados usando Putty.

Por último, en esta imagen (figura 32) se puede ver lo que ocurre al pisar el acelerador bruscamente. La velocidad sigue siendo nula, ya que el vehículo continúa estacionado, pero las revoluciones se disparan haciendo saltar un aviso de peligro al superar 4000rpm. También se puede observar que ahora la mezcla ya no es rica en combustible porque el motor necesita más potencia, según el cálculo instantáneo (Short Term Fuel Trim), y la posición del acelerador y el avance de sincronización de chispa han aumentado, aunque este último sigue siendo menor de lo esperado. Por

contra, el valor calculado de carga ha disminuido, ya que ahora el par disponible es menor.

También hay alertas programadas para la velocidad y para el exceso de temperatura del refrigerante, como se puede ver en el *Anexo 3.3*, pero por motivos de seguridad, no he podido obtener imágenes sobrepasando sus valores límite.

Una vez comprobado que el cable OBD-II funciona sin problemas, que los datos son leídos y tratados fácilmente, que conozco todos los comandos disponibles en el vehículo, y que el programa diseñado funciona correctamente, es el momento de pasar a utilizar la placa definitiva del proyecto, Custodium Tracker.

La finalidad de usar esta placa es utilizar la posibilidad de leer y tratar diferentes parámetros de un vehículo para complementar su función original de localización y envío de datos.

Como ya estaba familiarizado con este hardware y ya había realizado ensayos de localización, me dispuse a grabar mi programa en dicha placa. Para ello, lo primero fue incluir las librerías del cable (OBD.h y OBD.cpp) junto con todas las librerías que Custodium necesita en la carpeta en la que el programa iba a ser guardado. Después introduje mis líneas de código en el programa que la empresa me había proporcionado para su producto, respetando la estructura de programación de Arduino IDE e intercalando cada parte donde le correspondía para que el software lo interpretara correctamente, pero reduciendo el número de parámetros para que resultase más clara la información consultada. Omití los comandos de escritura, ya que con esta placa no se iban a visualizar los resultados utilizando el monitor serie como en el caso de Arduino. Finalmente, en el comando ya existente en el programa de la empresa para enviar los datos de fecha y GPS, tuve que realizar alguna pequeña modificación e introducir los valores obtenidos por el cable, para que también fueran enviados al servidor de consulta. Esta nueva información se introdujo en un apartado extra llamado “otros”, para no mezclarla con el resto de datos enviados.

Con el programa finalizado y después de comprobar la ausencia de errores con Arduino IDE, el siguiente paso era grabar el programa en la placa para realizar un primer ensayo, y comprobar así que la memoria RAM de Custodium Tracker era suficiente. Como ya se ha dicho, por motivos de espacio, esta placa no contiene ningún conector para poder grabar el programa, por tanto para ello es necesaria la utilización de un FTDI.

Una vez grabado, surgió el mismo problema de visualización que existía con Arduino, ya que Custodium Tracker solo dispone de un puerto serie y resulta imposible conectar el cable OBD-II y observar los resultados en el ordenador al mismo tiempo. Pero esta vez, debido a los pocos pines disponibles en esta placa, tampoco era posible configurar un puerto serie virtual. De manera que realicé primero un ensayo sin el cable y sin el

vehículo para comprobar que el programa seguía enviando los datos específicos de la placa (coordenadas GPS, fecha, hora, batería, etc), y poder ver en el ordenador, a través del monitor serie, tanto la memoria RAM disponible como los posibles errores.

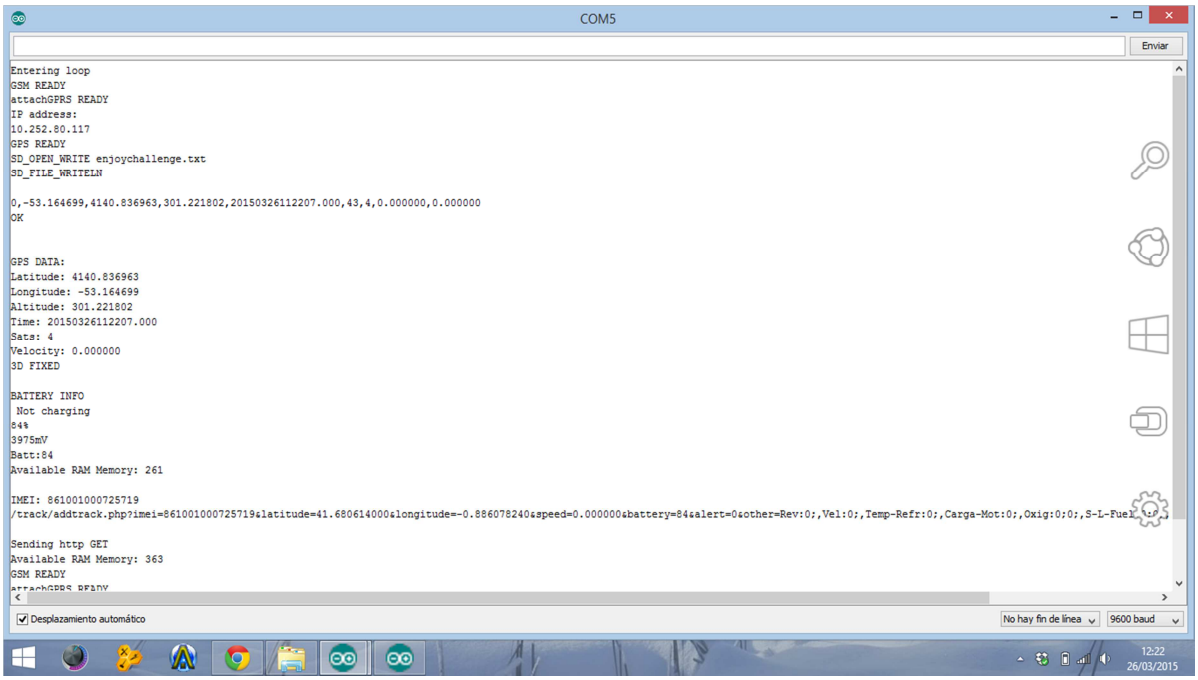


Figura 33. Monitor serie de Arduino IDE.

Después de ver que todo funcionaba correctamente, realicé un nuevo ensayo utilizando el cable y el vehículo en lugar del ordenador, ya que después de comprobar la ausencia de errores, ya no era necesario el monitor serie porque se podían consultar los datos en el servidor web.

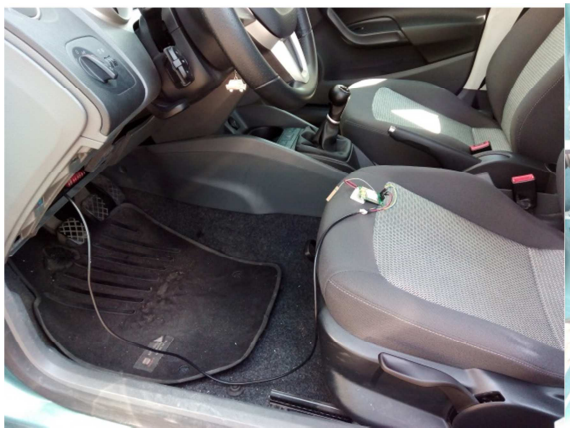


Figura 34. Conexión de Custodium Tracker al vehículo.

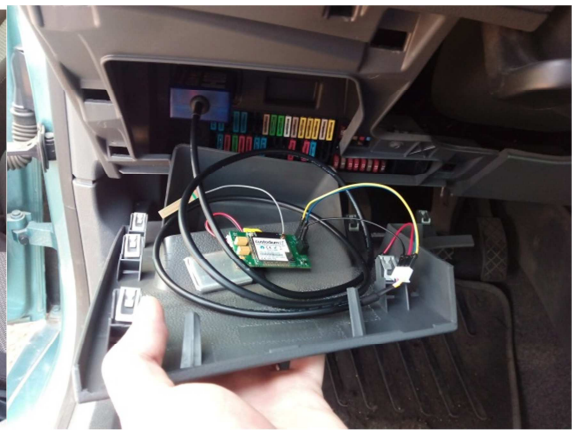


Figura 35. Conexión de Custodium Tracker al vehículo.

Fueron necesarios varios ensayos y correcciones para conseguir que funcionara perfectamente, ya que al principio los datos del vehículo eran enviados pero en el

servidor aparecían con valor cero. Finalmente, conseguí que se mostrasen los datos leídos por medio del cable y que pudieran ser consultados desde la página web o desde la aplicación móvil.

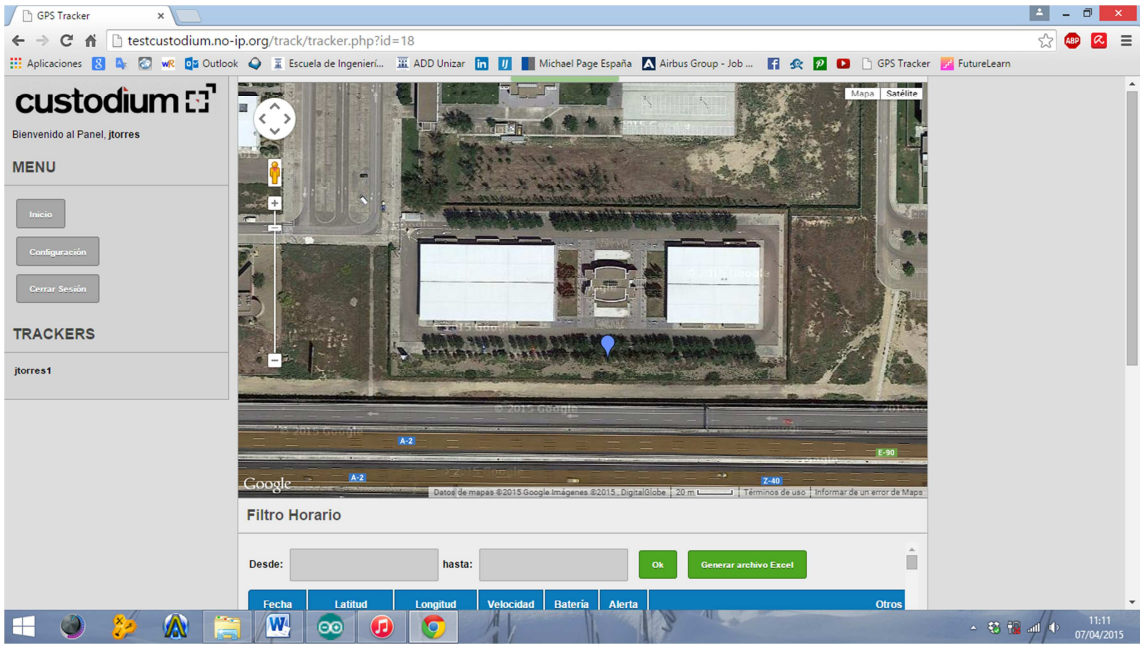


Figura 36. Servidor web de Custodium. Mapas.

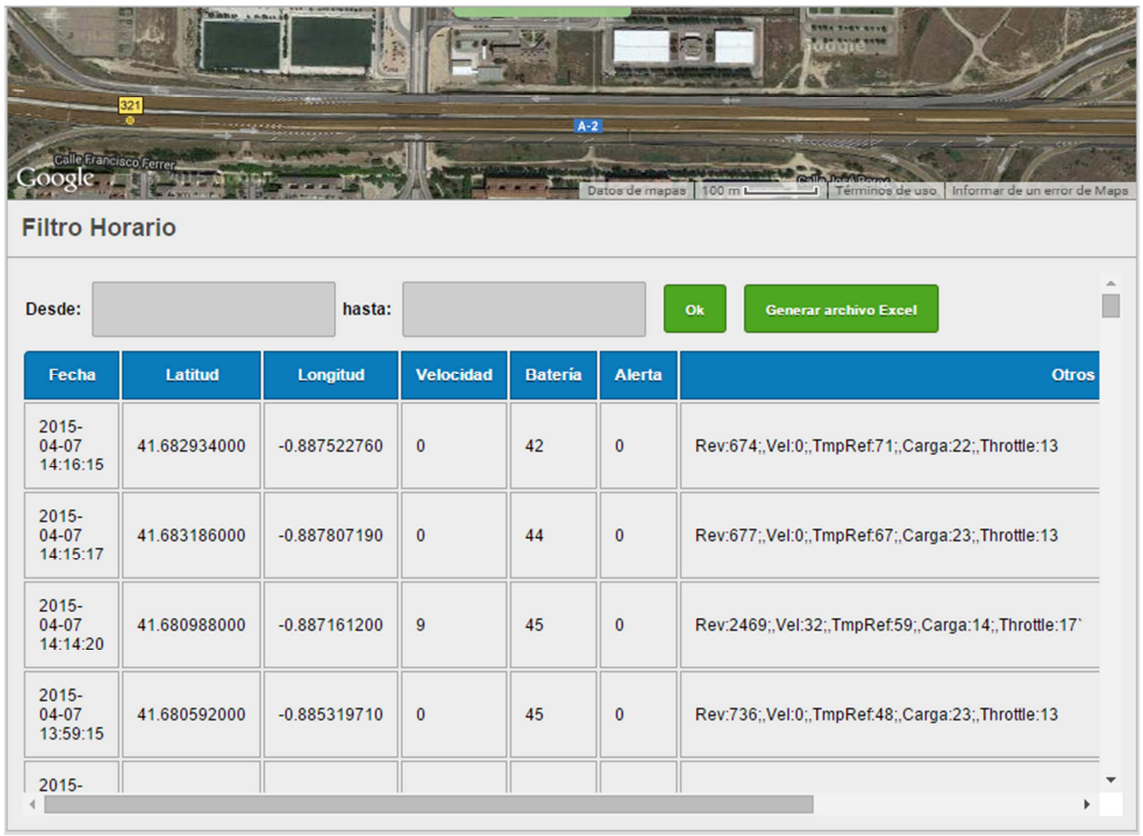


Figura 37. Servidor web de Custodium. Datos.

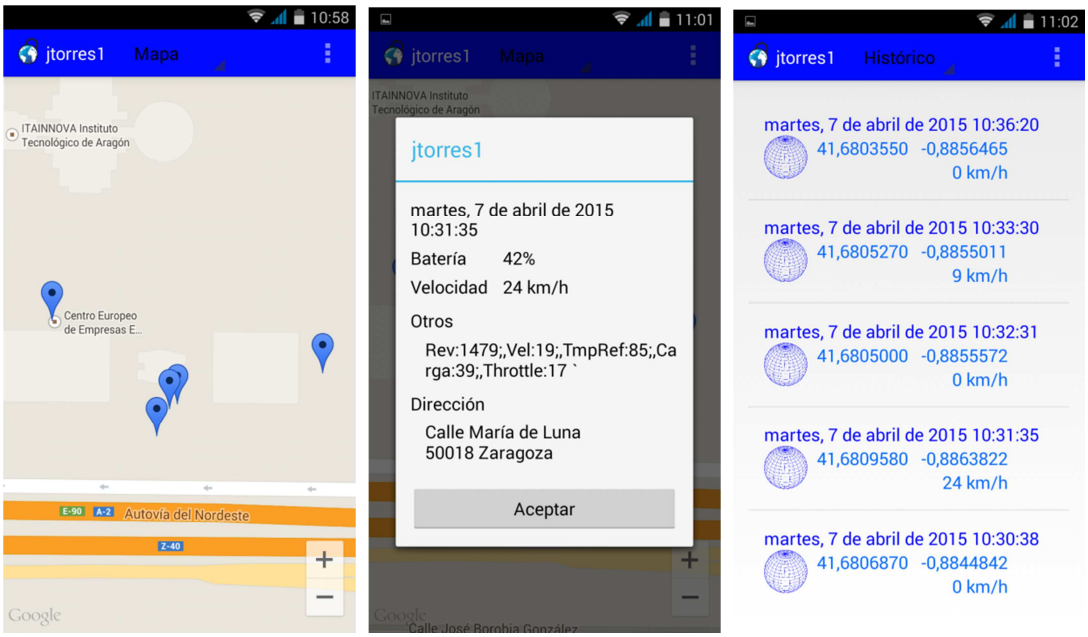


Figura 38. Aplicación móvil. Localización, datos de cada vehículo y registro histórico (respectivamente).

Como se puede observar, la información mostrada en ambos sitios es prácticamente la misma, pero es preferible utilizar la web vista en el ordenador, ya que dispone de alguna función más, los mapas se ven con más resolución, se pueden consultar todos los datos obtenidos sin importar su antigüedad y se visualizan en una pantalla mayor. Por todo ello, me centré únicamente en la página web.

Para que el sistema resultase más útil decidí introducir varias alertas, igual que había hecho con Arduino, pero en este caso no podía introducir ningún comentario explicativo, ya que la placa está programada para que solo envíe valores numéricos. Así que aprovechando el apartado ya existente reservado para la alerta de batería baja, introduje los avisos necesarios para comunicar, según los datos leídos, un fallo o uso indebido del vehículo.

Además de la alerta de batería, que avisaba cuando esta bajaba del 10%, introduje una para avisar cuando las revoluciones fueran superiores a 5000rpm, otra para comunicar una velocidad superior a 120km/h, otra para avisar si la temperatura del líquido refrigerante aumenta de 95ºC, y por último, una alerta imprescindible para evitar robos, que utilizando las coordenadas GPS, manda un aviso cuando el vehículo se aleja demasiado de su zona de uso permitido.

Pero el programa de Custodium estaba diseñado para enviar solamente un 1 cuando la batería fuese baja. Se planteó por tanto la cuestión de cómo introducir el resto de

alertas, cómo diferenciar una de otra, y finalmente cómo poder mostrar más de una al mismo tiempo, en el caso de que hiciesen falta varios avisos.

Lo primero fue diseñar las funciones que avisarían de un posible fallo al leer los parámetros. Al final de cada una de estas funciones, se incorporó el comando que asignaba un valor numérico a las alertas (alert= x;), del mismo modo que el utilizado para la batería pero eligiendo un número diferente para cada una de ellas. Finalmente, para que no se solapasen los valores cuando se activase más de una función, diseñé un método para poder mostrar varios de ellos al mismo tiempo. Cada una de las alertas multiplicaría por 10 el valor de alerta actual y le sumaría su propio número, de manera que enviando al servidor un único valor numérico, se podría saber cuántos avisos hay y cuáles son. Por ejemplo, si se observa un valor de 134, se puede saber que la primera, la tercera y la cuarta función han detectado un error y han avisado.

Todo esto debe introducirse en el programa antes del comando de envío de datos, para que se mande el valor final de alerta (Ver *Anexo 3.4*). Posteriormente, la persona encargada de identificar dichas alertas debe comprobar a qué error corresponde cada número, sabiendo que 1 se refiere a la alerta de batería baja, 2 es para la alerta de revoluciones, 3 indica velocidad excesiva, 4 avisa si la temperatura del líquido refrigerante es elevada y 5 quiere decir que el vehículo ha salido de una cierta zona.

Con este nuevo cambio en el programa, realicé más ensayos para comprobar que todo funcionaba, y así fue. Los datos se seguían enviando sin problemas, pero además, aparecían los números de cada una de las alertas cuando se superaba el límite programado. Para estos ensayos, bajé los valores de riesgo a 2000rpm, 20km/h y 80ºC para poder practicar sin peligro. También reduje la zona de uso permitido a los alrededores de la universidad EINA, donde se encuentra la empresa en la que he desarrollado el proyecto, de manera que al alejarme de ella pudiese ver si la alerta funciona correctamente.

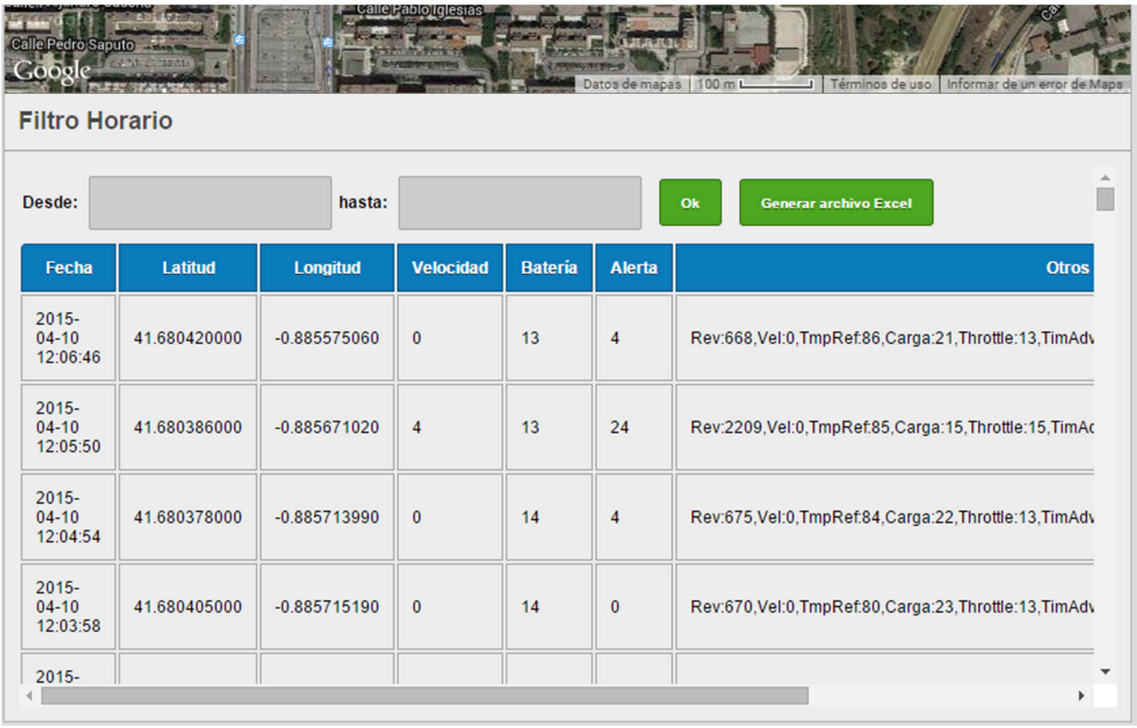


Figura 39. Servidor web de Custodium. Alertas.

Como se puede observar en esta imagen (figura 39), hasta que la temperatura del líquido refrigerante no supera los 80ºC, no aparece su alerta correspondiente (alerta 4). Y lo mismo ocurre con las revoluciones, cuyo aviso (alerta 2) no se muestra hasta que no superan un valor de 2000rpm, añadiéndose a la alerta ya existente.

También se puede observar la relación que guardan los parámetros, ya que al mantener levemente pisado el acelerador durante varios segundos para superar el límite de revoluciones, se puede ver como disminuye el porcentaje de carga disponible en el motor y aumenta ligeramente la posición del pedal.

Para demostrar la veracidad y exactitud de los datos se puede ver la imagen siguiente (figura 40), donde observando la hora del coche y las revoluciones, se puede comprobar que corresponde al instante en el que se supera el límite de la alerta 2.

Figura 40. Panel de instrumentos (Seat Ibiza).

Respecto a la velocidad, el programa tiene dos datos diferentes, el proporcionado por el GPS y el obtenido de la centralita del vehículo. Por diversas razones, este segundo siempre es superior al proporcionado por el GPS, que por ser calculado de manera ajena al vehículo, mostrará un valor bastante real. La variación experimentada en el Seat Ibiza con el que se ha desarrollado este proyecto está en torno al 6% (GPS: 50km/h; Centralita: 53km/h).

Un motivo evidente de esta diferencia son los neumáticos, ya que el vehículo calcula la velocidad en función del número de vueltas de la rueda por unidad de tiempo (medido a la salida de la caja de cambios), y dependiendo del desgaste, del hinchado y del modelo de neumático, este parámetro puede variar.

Como se puede ver en la página web que aparece en la bibliografía, donde se muestra la velocidad real en función de la teórica para cada neumático equivalente al de serie, la diferencia nunca será superior al 3%, ya que lo impide la legislación Española, y en algunos casos el contador marcará incluso menos velocidad de la real. Además de no ser suficiente para alcanzar la diferencia de velocidad experimentada, el Seat Ibiza utilizado lleva los neumáticos de serie, por tanto esta no es la razón.

Parece evidente que se deberá al desgaste o al inflado de ruedas, pero en este caso los neumáticos están en buen estado y ajustados a la presión que recomienda el vehículo.

La razón de esta variación parece ser, según la revista online “autopista.es”, una Directiva Europea que regula cómo deben ser los velocímetros de los automóviles que se comercializan en sus estados miembros, aparentemente por motivos de seguridad.

Esto genera una pequeña diferencia en los resultados medidos, pero además, como se puede observar al mirar la hora de los datos, el programa tarda aproximadamente un minuto en ejecutarse por completo y ambos comandos de velocidad están en partes distintas del código, lo que quiere decir que cada uno lee el valor en momentos distintos y si el vehículo no lleva una velocidad constante, la diferencia en los resultados puede ser mayor aún.

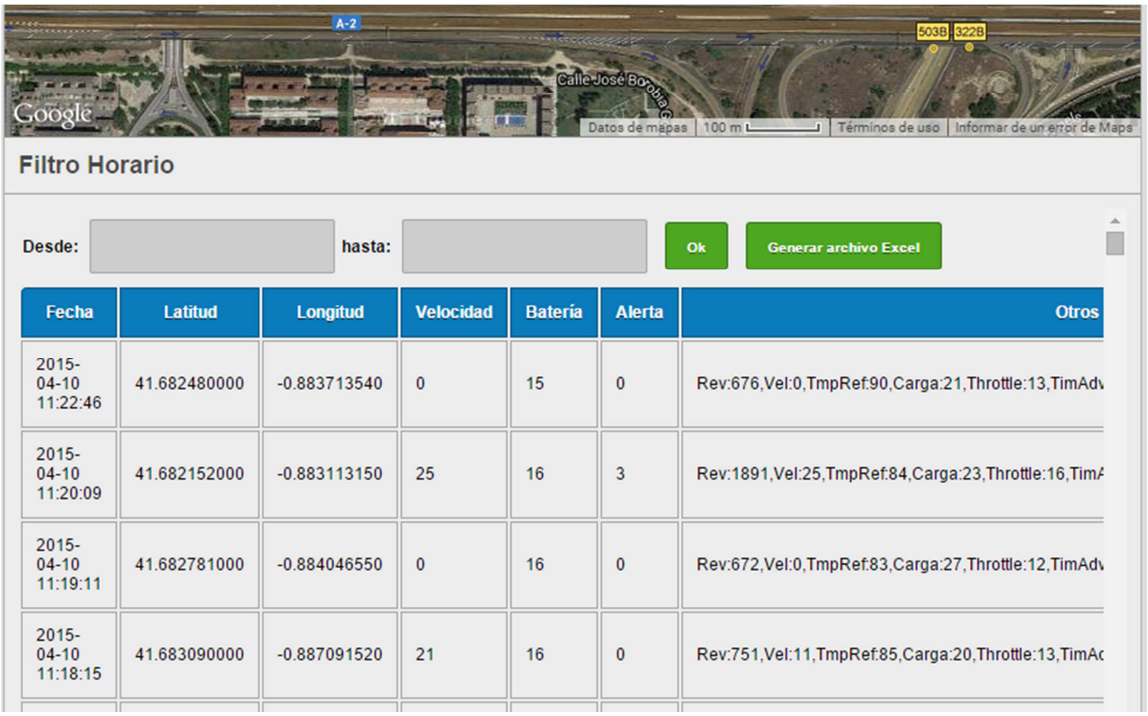


Figura 41. Servidor web de Custodium. Alertas.

En esta imagen (figura 41) se puede ver, que en el primer ensayo, a las 11:18, el GPS mostraba una velocidad de 21km/h, mientras que la lectura del vehículo era de 11km/h. Esta gran diferencia se debe a una velocidad irregular a lo largo de la ejecución del programa, pero debido a que la alerta está programada para los datos de la centralita, no aparece el aviso correspondiente para el límite de 20km/h.

En el siguiente ensayo, a las 11:20, manteniendo una velocidad constante y superior a la del caso anterior, se consigue evitar la diferencia entre ambos datos, y al superar dicho límite aparece la alerta.

Por último, en las siguientes imágenes se puede ver el correcto funcionamiento de la alerta de localización.

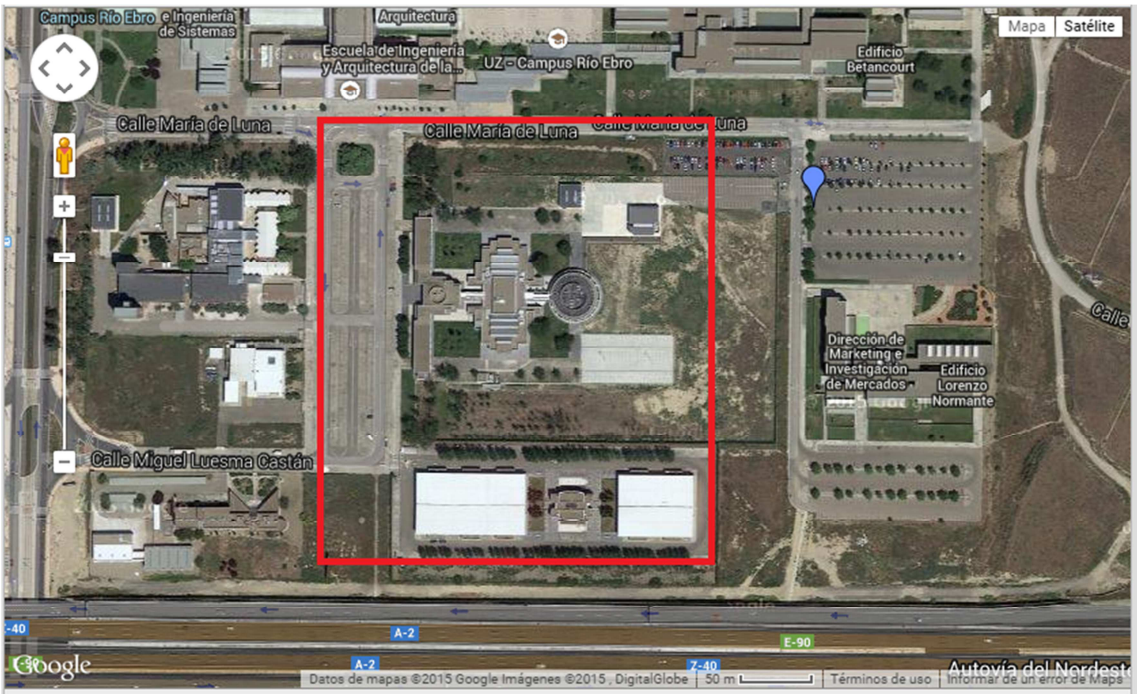
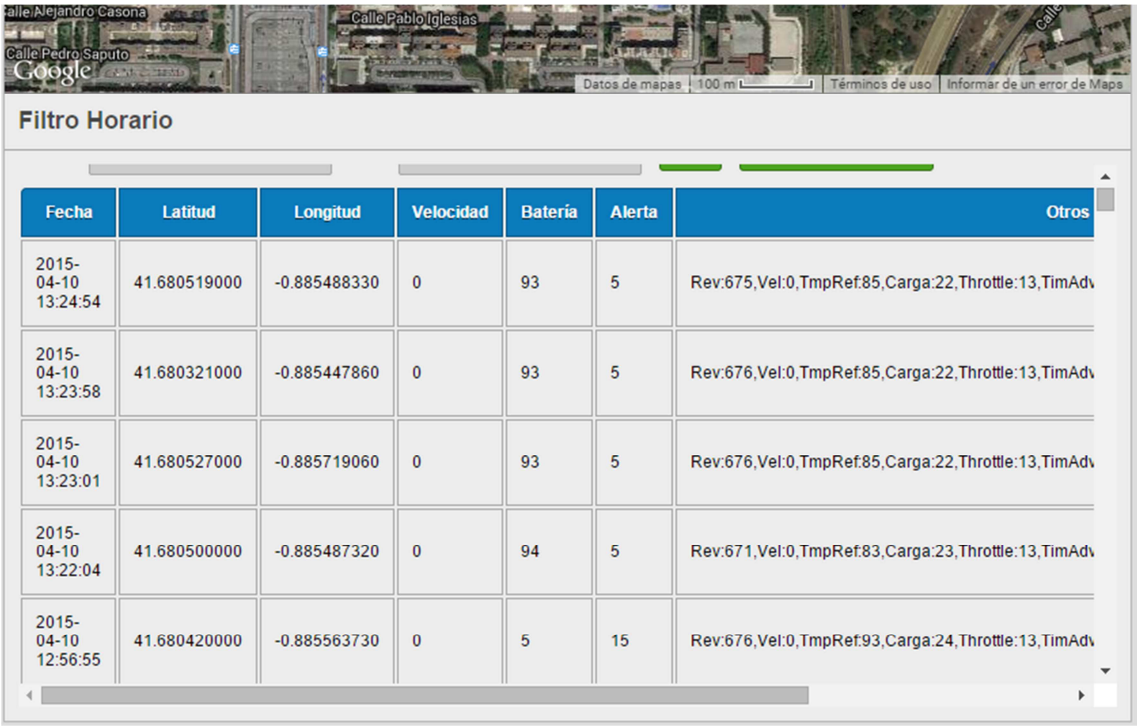


Figura 42. Localización del vehículo fuera de la zona de uso permitido.



Fecha	Latitud	Longitud	Velocidad	Batería	Alerta	Otros
2015-04-10 13:24:54	41.680519000	-0.885488330	0	93	5	Rev:675,Vel:0,TmpRef:85,Carga:22,Throttle:13,TimAdv
2015-04-10 13:23:58	41.680321000	-0.885447860	0	93	5	Rev:676,Vel:0,TmpRef:85,Carga:22,Throttle:13,TimAdv
2015-04-10 13:23:01	41.680527000	-0.885719060	0	93	5	Rev:676,Vel:0,TmpRef:85,Carga:22,Throttle:13,TimAdv
2015-04-10 13:22:04	41.680500000	-0.885487320	0	94	5	Rev:671,Vel:0,TmpRef:83,Carga:23,Throttle:13,TimAdv
2015-04-10 12:56:55	41.680420000	-0.885563730	0	5	15	Rev:676,Vel:0,TmpRef:93,Carga:24,Throttle:13,TimAdv

Figura 43. Servidor web de Custodium. Alertas.

En el primero de los ensayos también aparece la alerta de batería baja, que desaparece justo después al recargarla (figura 43).

A continuación, se me ocurrió que podría utilizar la fecha proporcionada por el GPS para obtener más resultados y ampliar el número de alertas enviadas. Esta fecha podría ser útil por ejemplo, para conocer los años de revisión del vehículo (ITV) o para calcular aproximadamente cuándo toca cambiar los neumáticos debido a su desgaste.

La normativa de revisiones ITV (Inspección Técnica de Vehículos) dice: que los turismos particulares de primera matriculación (que es el tipo de vehículo en el que se va a realizar el ensayo) están exentos de inspección durante los 4 primeros años, deben ser revisados cada 2 años entre los 4 y 10 años de antigüedad y deben pasar la inspección cada año de ahí en adelante.

Para los neumáticos, no existe una normativa estricta en cuanto al tiempo de uso, pero se recomienda cambiarlos cada 5 años o 40000 km, lo que antes ocurra, revisándolos continuamente para detectar grietas anómalas y comprobar que la profundidad del dibujo no es inferior al límite legal de 1.6 mm.

Dado que el parámetro de la distancia recorrida no es visible a través del cable OBD-II, solo se puede avisar de estos dos eventos usando un medidor de tiempo, así que creé dos nuevas funciones e introduje en ellas el año proporcionado por la fecha del GPS, ya que es la unidad de tiempo en que se miden dichos avisos.

Para saber cuándo se ha cumplido un período de tiempo, se necesita un valor de referencia que se pueda comparar con el proporcionado por el GPS. Para que este valor no se pierda si falla la alimentación de la placa, debe ser guardado en la memoria EEPROM de Custodium Tracker, y para ello se debe crear una nueva variable de tipo “byte”, que es el utilizado por este tipo de memoria. A esta variable se le dará el valor del año en el que se empieza a contar, pero como el tipo de dato byte solo soporta números hasta el 256, al valor de los años hay que restarle 2000 (2015 → 15).

Usando este método, el programa avisará de la tarea a realizar el 1 de enero del año correspondiente y deberá actualizarse manualmente el año de referencia para que la alerta se detenga y empiece de nuevo la cuenta. Para ello, una vez realizada la tarea de inspección o cambio de neumáticos, se tendrá que conectar el ordenador a la placa y grabar de nuevo el mismo programa pero incluyendo unos comandos concretos que actualicen el valor de la memoria EEPROM. Dichos comandos, que se encuentran siempre desactivados (ver *Anexo 3.4*), solamente se incluyen en este caso concreto y deben volverse a desactivar a continuación para volver a grabar en la placa el código de siempre. De esta manera, solo se actualiza el valor una vez y el programa puede seguir repitiéndose con normalidad otro período de años.

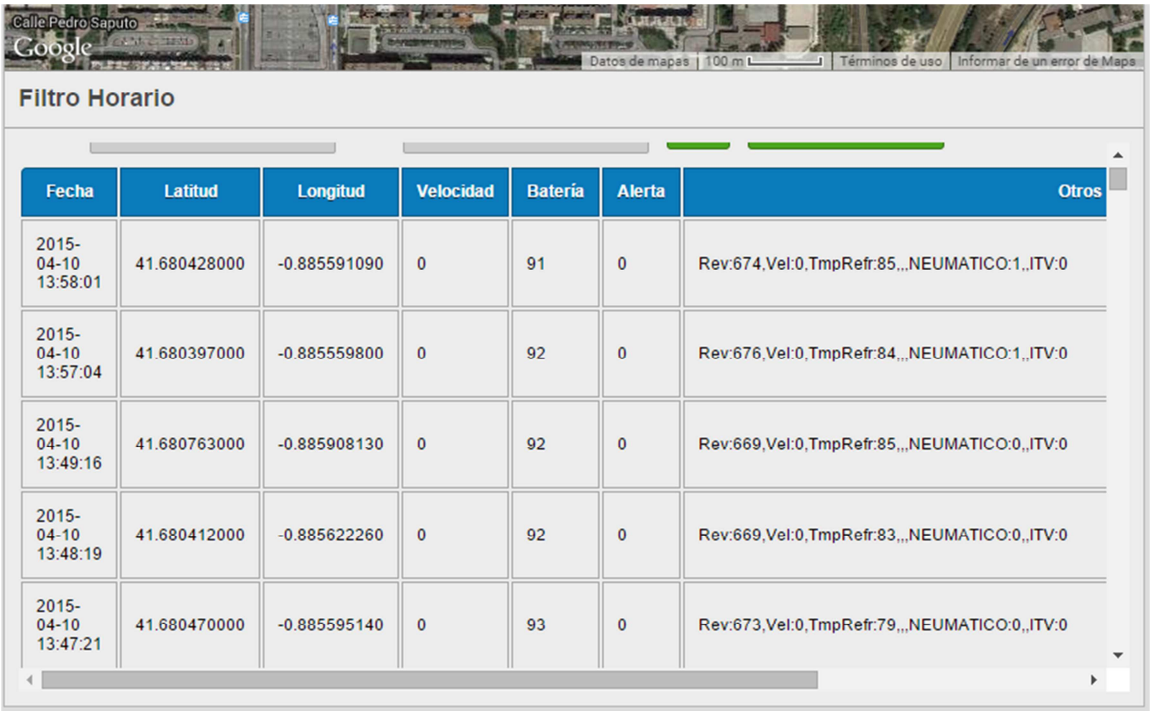
Para poder reiniciar solo una de las tareas y no modificar el contador de la otra, son necesarias dos variables diferentes que se guardan en la memoria EEPROM con distintos años de referencia (year1, year2).

Ya que estos avisos no son tan urgentes como otros y se pueden corregir a lo largo de todo un año, no los introduje en el apartado de alertas. En lugar de ello, ambas variables aparecen en el apartado “otros” y enviarán al servidor un valor de 1 cuando haga falta alguna tarea o un valor de 0 cuando no sea necesario.

Con estas nuevas funciones introducidas y después de verificar la ausencia de errores básicos, realicé varios ensayos para comprobar que todo funcionaba perfectamente. Primero hice una prueba usando el ordenador, para ver si la memoria RAM seguía siendo suficiente, para verificar que no había fallos de funcionamiento y para comprobar si el GPS proporcionaba el año correcto. A continuación realice varios ensayos con el cable y el vehículo, modificando manualmente los años para ver si se producían los avisos que se tendrán que producir después de largos períodos de tiempo.

Para ello introduje los comandos “EEPROM.write(0,10)” y “EEPROM.write(1,10)”, lo cual modifica las dos fechas de referencia utilizadas y le indica al programa que la puesta a cero se hizo en el año 2010. Ya que nos encontramos en el año 2015, el aviso de neumáticos debería activarse por haberse superado los 4 años, pero el de ITV no debería activarse porque no hay ninguna revisión a los 5 años (ver funciones al final del *Anexo 3.4*).

Todo funcionó correctamente, así que el programa estaba terminado.



Fecha	Latitud	Longitud	Velocidad	Batería	Alerta	Otros
2015-04-10 13:58:01	41.680428000	-0.885591090	0	91	0	Rev:674,Vel:0,TmpRefr:85,,,NEUMATICO:1,,ITV:0
2015-04-10 13:57:04	41.680397000	-0.885559800	0	92	0	Rev:676,Vel:0,TmpRefr:84,,,NEUMATICO:1,,ITV:0
2015-04-10 13:49:16	41.680763000	-0.885908130	0	92	0	Rev:669,Vel:0,TmpRefr:85,,,NEUMATICO:0,,ITV:0
2015-04-10 13:48:19	41.680412000	-0.885622260	0	92	0	Rev:669,Vel:0,TmpRefr:83,,,NEUMATICO:0,,ITV:0
2015-04-10 13:47:21	41.680470000	-0.885595140	0	93	0	Rev:673,Vel:0,TmpRefr:79,,,NEUMATICO:0,,ITV:0

Figura 44. Servidor web de Custodium. Avisos de ITV y neumáticos.

El código final, resultado de la unión del programa de Custodium, de mi programa y de ciertas modificaciones añadidas, como alertas y comandos para GPS, se puede ver en el *Anexo 3.4*, pero por motivos de confidencialidad no se muestra el código completo perteneciente a la empresa. Si fuera necesario se podría mostrar mediante previa autorización de Custodium.

Llegado a este punto, la viabilidad del proyecto queda demostrada, ya que se ha diseñado la herramienta de lectura, cálculo y comunicación que se buscaba obtener desde un primer momento. Pero a pesar de la localización GPS y de haber incluido parámetros de velocidad, temperatura y tiempo, el resto de comandos siguen siendo más adecuados para la optimización de vehículos que para su gestión y control a distancia. Por ello, utilizando este código, las mismas herramientas y los conocimientos adquiridos, he diseñado otro programa que contiene parámetros más interesantes, con la esperanza de que futuras ampliaciones de este proyecto, como las propuestas en el *Anexo 5*, permitan su lectura y proporcionen ese valioso instrumento de gestión que puede ser tan beneficioso para empresas con grandes flotas de vehículos.

Partiendo por tanto del programa ya realizado, he sustituido los valores relacionados con el funcionamiento del motor, por otros más útiles obtenidos también de la tabla del *Anexo 1*. Estos son: el tiempo de funcionamiento del motor, la distancia recorrida, el nivel de combustible, la temperatura ambiente, la temperatura del aceite y el número de averías del vehículo. Además de estos parámetros que no necesitan ninguna explicación, se conservan otros del programa anterior, como la velocidad, las revoluciones y la temperatura del líquido refrigerante. También permanecen las funciones de ITV y cambio de neumáticos, que además de utilizar los años del GPS, ahora también utilizan los km recorridos. Y he añadido otras funciones nuevas para calcular la cantidad de combustible rellenado, el número de recargas y varias alertas que avisan al alcanzar nuevos valores límite. Para no perder los datos de contadores o variables si falla la alimentación por alguna razón, estos valores, igual que en el programa anterior, se guardan en la memoria EEPROM de Custodium Tracker. Este nuevo programa se puede ver en el *Anexo 3.5*.

Por el momento, los nuevos parámetros dan un valor nulo, como cabía esperar, pero en el caso de encontrar alguna solución para su lectura, se podrían realizar ensayos utilizando este nuevo código.

5.3. SEGUIMIENTO GPS

Para terminar, hay que comprobar si los datos proporcionados por el GPS son siempre bastante exactos y si podrían resultar útiles en caso de robo.

Para ello, conecté el cable y la placa al vehículo (Seat Ibiza) con el programa del *Anexo 3.4* grabado, y después de ver que el servidor recibía sin problemas los datos de GPS y funcionamiento del coche, realicé varios recorridos, deteniéndome cada cierto tiempo para comprobar que la ubicación coincidía con la información obtenida.

Ya se había verificado anteriormente la alerta de localización, aunque eso solo indicaba que el GPS y el programa funcionan bien utilizando unas coordenadas aproximadas, pero todavía no se había comprobado la precisión del sistema.

Figura 45. Seat Ibiza junto al edificio Torres Quevedo (EINA).

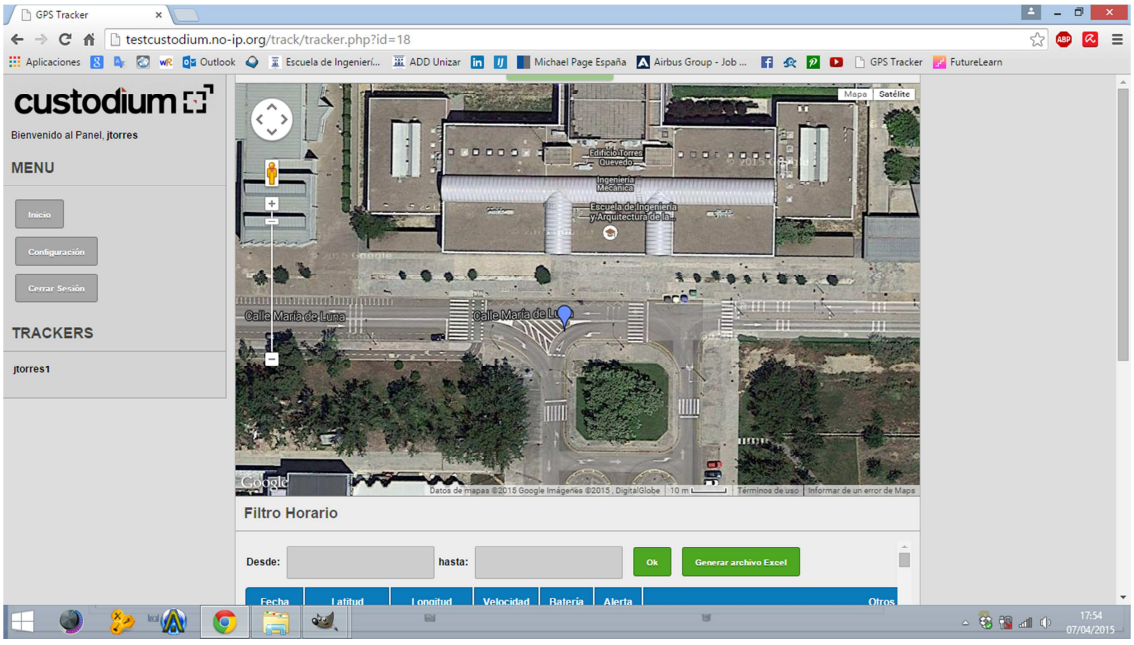


Figura 46. Servidor web de Custodium. Localización del vehículo junto al edificio Torres Quevedo (EINA).

Figura 47. Seat Ibiza junto al edificio central del CEEI Aragón.

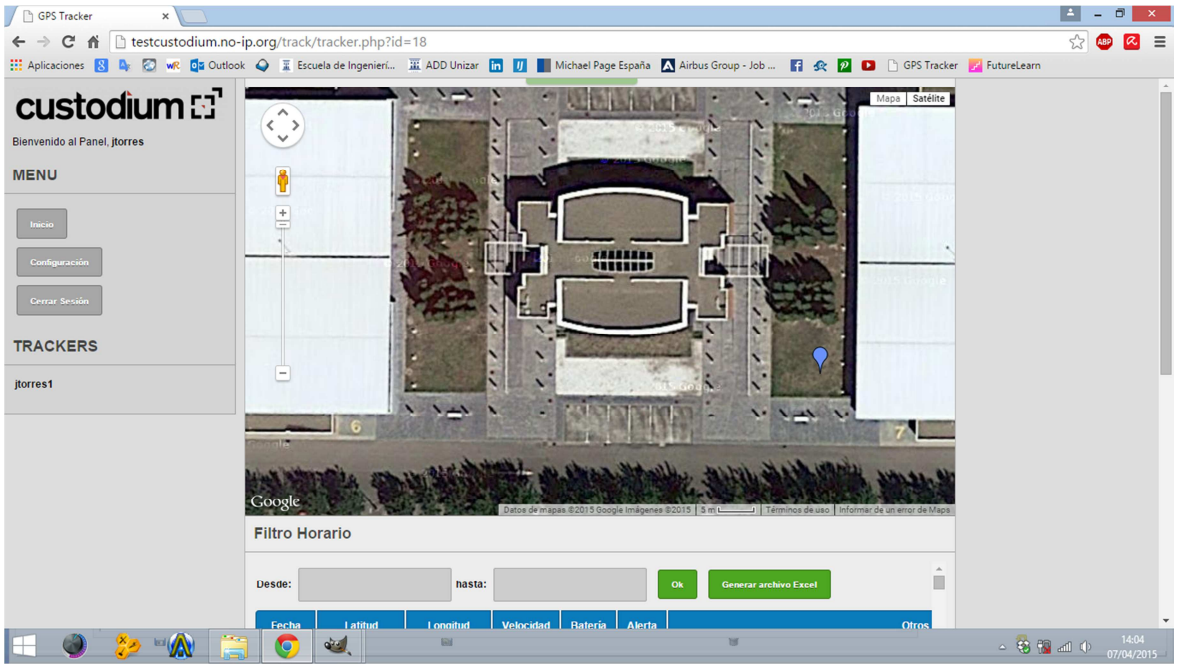


Figura 48. Servidor web de Custodium. Localización del vehículo junto al edificio central del CEEI Aragón.

Viendo estas imágenes se puede apreciar un margen de error de 5-10 metros, pero el sistema funciona correctamente.

Observando los datos también se puede saber cuándo se ha subido una pendiente a lo largo de un recorrido, ya que ello genera un valor alto de revoluciones, un porcentaje positivo de los “Fuel Trim”, un voltaje bajo de los sensores de oxígeno y un incremento moderado de la posición del acelerador. Todo esto conlleva un aumento en el consumo de combustible, que aunque no se pueda leer directamente del vehículo, se podría prever analizando el resto de la información.

Realizando muchos recorridos distintos y comparando los datos obtenidos en ellos, podrían planificarse rutas que gastasen menos combustible debido a las condiciones del terreno, lo que generaría enormes ahorros en empresas de reparto o transporte de mercancías, en las que solo importa el destino y no el camino elegido.

6. CONCLUSIONES

Siguiendo el plan previsto, se han alcanzado los objetivos fijados, diseñando una herramienta muy útil para gestionar empresas con grandes flotas de vehículos.

Este sistema permite a la persona responsable obtener desde una oficina, únicamente con la necesidad de una conexión a internet, datos de todos sus vehículos. Estos datos se pueden usar como ya se ha visto, para evitar costes excesivos de consumo de combustible, multas por exceso de velocidad o retraso en la ITV, posibles averías por desconocimiento de un error de funcionamiento, evitar peligros para los conductores por mal estado del vehículo, conocer el uso diario que los empleados hacen de ellos para evitar fraudes a la empresa o permitir la detección y localización GPS en el caso de que alguno sea robado.

Para ello, solo hay que instalar en cada uno de los vehículos este sistema formado por placa y cable de lectura. El proceso de instalación es muy sencillo, ya que únicamente hay que conectar el cable en el puerto OBD-II y tres pines del cable a la placa Custodium. Además, este sistema es un prototipo, que en el caso de tener éxito o comercializarse, se podría simplificar todavía más.

Su uso diario también es muy simple, ya que solamente es necesario conectarse al servidor web usando el nombre de usuario y la contraseña, observar si existe alguna alerta y consultar los valores que se desee para observar el funcionamiento de cada uno de los vehículos.

Lo más difícil de este sistema sería reiniciar los contadores de determinadas alertas, pero en el código mostrado en los anexos, ya aparecen los comandos que se necesitan para ello y su ejecución puede ser realizada por cualquier persona que tenga unos

conocimientos básicos de electrónica, siguiendo las simples instrucciones citadas anteriormente en el proyecto.

Este sistema también puede tener otras utilidades muy interesantes alejadas de la idea inicial de gestión de flotas, pero algunas necesitarían una mejora en la lectura de datos y una ampliación o modificación del código utilizado aquí.

Una de ellas podría ser por ejemplo, la utilización de la información obtenida para realizar estudios de mercado o diseñar mejoras según el uso que la gente hace de los vehículos. Si se piensa por ejemplo en una empresa de fabricación de coches, podría resultarle interesante una herramienta que muestre con total claridad las deficiencias o diseños exitosos de su producto una vez que este se encuentre en el mercado, o el uso que hacen del coche los clientes que lo compran, con el fin de orientar la empresa hacia un rumbo determinado, realizando unas mejoras u otras. Aunque esto supondría un problema de privacidad de datos y habría que solicitar una autorización del cliente incentivada por un descuento en el precio.

También podría resultar interesante para empresas de seguros, ya que gracias a la información obtenida, sabrían cuáles de sus clientes podrían beneficiarse de un descuento por suponer un riesgo menor y en qué situaciones deben hacerse responsables de un accidente.

Otra función muy útil ya nombrada en el proyecto, sería el uso de los datos para la creación de rutas más eficientes en términos de combustible, tiempo u otros criterios, lo cual podría resultar interesante para empresas de reparto o de transporte y para fabricantes de mapas de carreteras o guías turísticas.

Por supuesto también ayudaría a cualquier empresa que necesite prevención de robos y localización de vehículos, como podría ser el caso de empresas de alquiler o de taxis.

En definitiva, se trata de un novedoso sistema de lectura, análisis y envío de datos a distancia, que junto con un módulo de localización GPS, crea una valiosa herramienta de tamaño muy reducido con poca o ninguna competencia en el mercado (Ver *Anexo 4*).

7. ANEXOS

7.1. ANEXO 1: PARÁMETROS DE COMUNICACIÓN

PID	Descripción
0x00	Supported PIDs 0x01 to 0x1F
0x01	Number of trouble codes, MIL indicator on/off, and available on-board tests
0x02	Freeze frame DTC
0x03	Fuel system status
0x04	Calculated engine load value
0x05	Engine coolant temperature
0x06	Short term fuel % trim - Bank 1
0x07	Long term fuel % trim - Bank 1
0x08	Short term fuel % trim - Bank 2
0x09	Long term fuel % trim - Bank 2
0x0A	Fuel pressure
0x0B	Intake manifold pressure
0x0C	Engine RPM
0x0D	Vehicle speed
0x0E	Timing advance
0x0F	Intake air temperature
0x10	Mass air flow rate
0x11	Throttle position
0x12	Secondary air status
0x13	Oxygen sensors present
0x14	Oxygen sensor voltage, Short term fuel trim (Bank 1, Sensor 1)
0x15	Oxygen sensor voltage, Short term fuel trim (Bank 1, Sensor 2)
0x16	Oxygen sensor voltage, Short term fuel trim (Bank 1, Sensor 3)
0x17	Oxygen sensor voltage, Short term fuel trim (Bank 1, Sensor 4)
0x18	Oxygen sensor voltage, Short term fuel trim (Bank 2, Sensor 1)
0x19	Oxygen sensor voltage, Short term fuel trim (Bank 2, Sensor 2)
0x1A	Oxygen sensor voltage, Short term fuel trim (Bank 2, Sensor 3)
0x1B	Oxygen sensor voltage, Short term fuel trim (Bank 2, Sensor 4)
0x1C	OBD standards this vehicle conforms to
0x1D	Oxygen sensors present 2
0x1E	Auxiliary input status
0x1F	Run time since engine start
0x20	Supported PIDs 0x21 to 0x3F
0x21	Distance traveled with malfunction indicator lamp (MIL) on
0x22	Fuel rail pressure (relative to manifold vacuum)
0x23	Fuel rail pressure (diesel)
0x24	Oxygen sensor 1 equivalence ratio (lambda value)
0x25	Oxygen sensor 2 equivalence ratio (lambda value)
0x26	Oxygen sensor 3 equivalence ratio (lambda value)
0x27	Oxygen sensor 4 equivalence ratio (lambda value)
0x28	Oxygen sensor 5 equivalence ratio (lambda value)
0x29	Oxygen sensor 6 equivalence ratio (lambda value)
0x2A	Oxygen sensor 7 equivalence ratio (lambda value)
0x2B	Oxygen sensor 8 equivalence ratio (lambda value)

0x2C	Commanded EGR
0x2D	EGR error
0x2E	Commanded evaporative purge
0x2F	Fuel level input
0x30	Number of warm-ups since codes cleared
0x31	Distance traveled since codes cleared
0x32	Evap system vapor pressure
0x33	Barometric pressure
0x34	Oxygen sensor 1 equivalence ratio (lambda value)
0x35	Oxygen sensor 2 equivalence ratio (lambda value)
0x36	Oxygen sensor 3 equivalence ratio (lambda value)
0x37	Oxygen sensor 4 equivalence ratio (lambda value)
0x38	Oxygen sensor 5 equivalence ratio (lambda value)
0x39	Oxygen sensor 6 equivalence ratio (lambda value)
0x3A	Oxygen sensor 7 equivalence ratio (lambda value)
0x3B	Oxygen sensor 8 equivalence ratio (lambda value)
0x3C	Catalyst temperature (Bank 1 Sensor 1)
0x3D	Catalyst temperature (Bank 2 Sensor 1)
0x3E	Catalyst temperature (Bank 1 Sensor 2)
0x3F	Catalyst temperature (Bank 2 Sensor 2)
0x40	Supported PIDs 0x41 to 0x5F
0x41	Monitor status this drive cycle
0x42	Control module voltage
0x43	Absolute load value
0x44	Command equivalence ratio
0x45	Relative throttle position
0x46	Ambient air temperature
0x47	Absolute throttle position B
0x48	Absolute throttle position C
0x49	Accelerator pedal position D
0x4A	Accelerator pedal position E
0x4B	Accelerator pedal position F
0x4C	Commanded throttle actuator
0x4D	Time run with MIL on
0x4E	Time since trouble codes cleared
0x4F	External test configuration #1
0x50	External test configuration #2
0x51	Fuel type
0x52	Percentage of alcohol fuel mix
0x53	Absolute evap system vapor pressure
0x54	Evap system vapor pressure
0x55	Short term secondary oxygen sensor trim bank 1 and bank 3
0x56	Long term secondary oxygen sensor trim bank 1 and bank 3
0x57	Short term secondary oxygen sensor trim bank 2 and bank 4
0x58	Long term secondary oxygen sensor trim bank 2 and bank 4
0x59	Fuel rail pressure (absolute)
0x5A	Relative accelerator pedal position
0x5B	Hybrid battery pack remaining life
0x5C	Engine oil temperature

0x5D	Fuel injection timing
0x5E	Fuel rate
0x5F	Emission requirements for this vehicle

7.2. ANEXO 2: CARACTERÍSTICAS DE LAS PLACAS

7.2.1. ARDUINO UNO

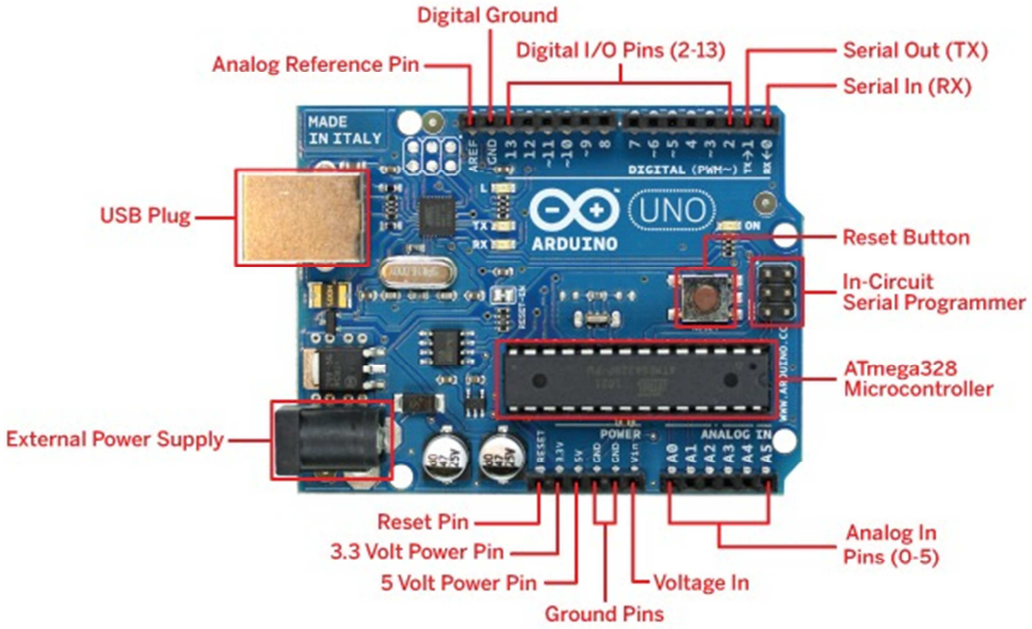


Figura 49. Esquema y características de Arduino UNO.

Microcontrolador	ATmega328
Voltaje de funcionamiento	5V
Voltaje de entrada (recomendado)	7-12V
Voltaje de salida (recomendado)	6-20V
Pines digitales I/O	14 (de los cuales 6 proporcionan salida PWM)
Pines analógicos de entrada	6
Corriente DC por pin I/O	40 mA
Corriente DC por pin de 3.3V	50 mA
Memoria Flash	32KB (ATmega328), 0.5KB usados por el bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad de reloj (Clock)	16 MHz
Longitud	68.6 mm
Anchura	53.4 mm
Peso	25 g

7.2.2. CUSTODIUM TRACKER

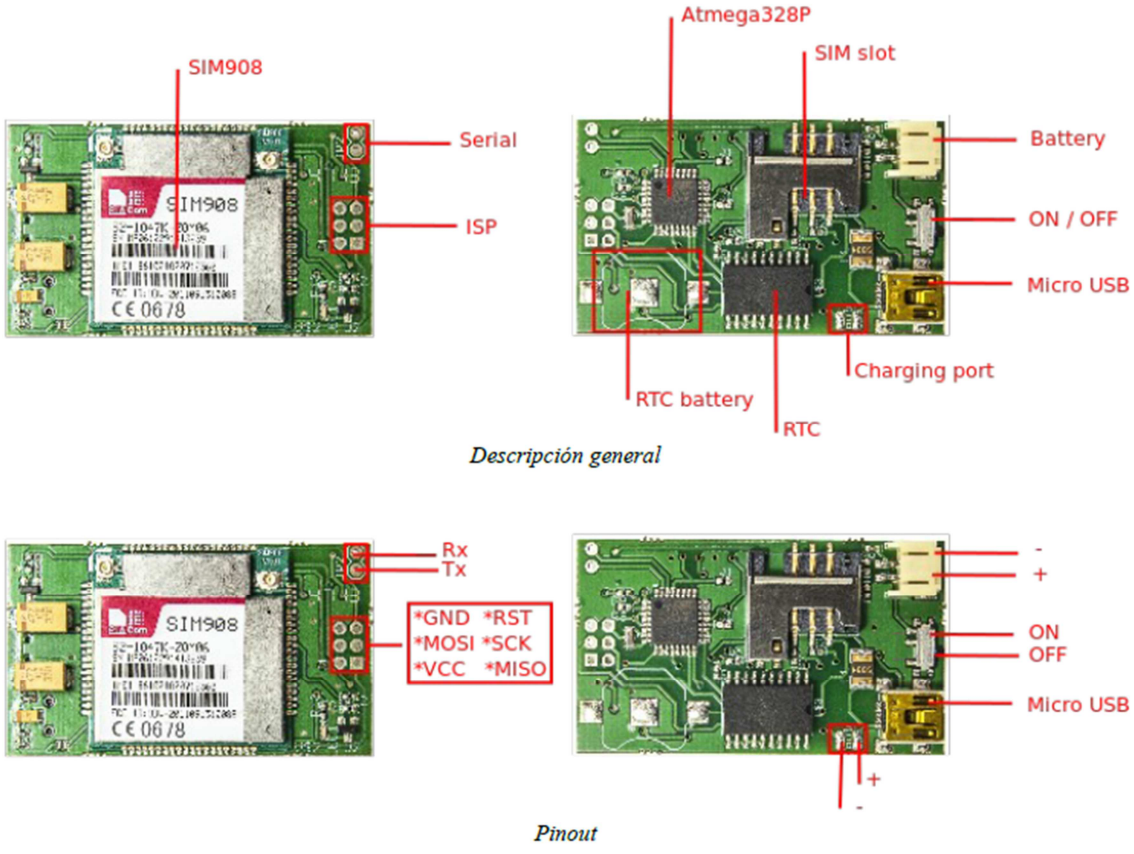


Figura 50. Esquema y características de Custodium Tracker.

- Microcontrolador Atmega328P a 8mhz
- Reloj en tiempo real (RTC) con batería de reserva
- Alojamiento para tarjeta SIM
- Interruptor de encendido/apagado
- Puerto mini USB para carga de batería
- Conector tipo JST para batería LiPo de 3,7V y 850mAh
- Módulo de comunicaciones GSM/GPRS/GPS SIM908
- Conectores para antena GSM y GPS
- Conector ISP (In-System Programming)
- Puerto serie
- Dimensiones: 21.5mm x 54mm

7.3. ANEXO 3: CÓDIGO USADO

7.3.1. Programa de prueba OBD-II (revoluciones-LED13)

```
/*  Este programa lee las revoluciones de un vehículo a través del cable OBD-II y
  enciende el LED del pin 13 cuando estas son mayores de 2000.  */

//Se incluyen las librerías necesarias para el programa.
#include <Wire.h>
#include "OBD.h"
COBD obd;  /* para el Modelo A (versión UART) */

void setup()
{
pinMode(13,OUTPUT);
obd.begin();    //Se inicia el cable OBD-II.
while (!obd.init());    //El programa permanece en este paso hasta que la conexión OBD-II esté operativa.
}

void loop()
{
  int value;   //Se inicializa la variable a utilizar.
  obd.read(PID_RPM, value);     //Se leen las revoluciones.
 //Se realiza una comparación del valor leído y se enciende o apaga el LED en función del resultado.
  if (value > 2000)
  {
    digitalWrite(13,HIGH);
  }
  else
  {
    digitalWrite(13, LOW);
  }
}
```

7.3.2. Primer programa de lectura y escritura de datos con Arduino

```
/*   Este programa lee varias variables de un vehículo a través de un cable OBD-II
  y las escribe por puerto serie en un ordenador, a través de un FTDI, usando
  Putty principalmente.  */

//Se incluyen las librerías necesarias para el programa.
#include <Wire.h>
```

```
#include "OBD.h"
#include <SoftwareSerial.h>
COBD obd;  /* para el Modelo A (versión UART) */

//Se crean un puerto serie virtual.
SoftwareSerial mySerial(3, 4); // RX, TX

//Se inicializan las variables a utilizar.
  int value1;
  int value2;
  int value3;
  int value4;
  int value5;
  int value6;
  int value7;
  int value8;

void setup()
{
  mySerial.begin(9600);  //Se abre el puerto serie a 9600 baudios
  mySerial.println("Iniciando OBD...");
  delay(500);
  obd.begin();    //Se inicia el cable OBD-II.
  while (!obd.init());    //El programa permanece en este paso hasta que la conexión OBD-II esté
operativa.
  delay(500);
  mySerial.println("OBD Iniciado");
}

void loop()
{
  obd.read(PID_RUNTIME, value1);    //Se lee la variable
  mySerial.print("Tiempo 1 (sec): ");  //Se escribe el texto
  mySerial.println(value1);              //Se escribe el valor

  obd.read(PID_TIME_WITH_MIL, value7);
  mySerial.print("Tiempo 2 (sec): ");
  mySerial.println(value7);

  obd.read(PID_TIME_SINCE_CODES_CLEARED, value8);
  mySerial.print("Tiempo 3 (sec): ");
  mySerial.println(value8);

  obd.read(PID_RPM, value2);
  mySerial.print("Revoluciones (rpm): ");
  mySerial.println(value2);

  obd.read(PID_SPEED, value3);
  mySerial.print("Velocidad (km/h): ");
  mySerial.println(value3);

  obd.read(PID_DISTANCE, value4);
```

```
  mySerial.print("Distancia (km): ");
  mySerial.println(value4);

  obd.read(PID_FUEL_LEVEL, value5);
  mySerial.print("Nivel de combustible (%): ");
  mySerial.println(value5);

  obd.read(PID_AMBIENT_TEMP, value6);
  mySerial.print("Temperatura ambiente (ºC): ");
  mySerial.println(value6);

  delay(2000);
}
```

7.3.3. Programa definitivo de lectura y escritura de datos/alertas con Arduino

```
/*    Este programa lee varias variables de un vehículo a través de un cable OBD-II
  y las escribe por puerto serie en un ordenador, así como determinadas alertas,
  usando Putty principalmente.    */

//Se incluyen las librerías necesarias para el programa.
#include <Wire.h>
#include "OBD.h"
#include <SoftwareSerial.h>
COBD obd;   /* para el Modelo A (versión UART) */

//Se crean un puerto serie virtual.
SoftwareSerial mySerial(3, 4); // RX, TX

//Se inicializan las variables a utilizar.
  int value1;
  int value2;
  int value3;
  int value4;
  int value5;
  int value6;
  int value7;
  int value8;
  int value9;
  int value10;
  int value11;
  int value12;
```

```
void setup()
{
  mySerial.begin(9600);   //Se abre el puerto serie a 9600 baudios, que es la velocidad a la que manda el cable OBD-II.
  mySerial.println("Iniciando OBD...");
  delay(500);
  obd.begin();  //Se inicia el cable OBD-II.
  while (!obd.init());  //El programa permanece en este paso hasta que la conexión OBD-II esté operativa.
  delay(1000);
  mySerial.println("OBD iniciado");
}

void loop()
{
  obd.read(PID_RPM, value1);   //Lee revoluciones (rpm)
  mySerial.print("Revoluciones (rpm): ");
  mySerial.print(value1);
  if (value1 > 4000)
  {
    mySerial.print("     ¡LIMITE EXCEDIDO!");
  }
  mySerial.println("");

  obd.read(PID_SPEED, value2);   //Lee velocidad (km/h)
  mySerial.print("Velocidad (km/h): ");
  mySerial.print(value2);
  if (value2 > 120)
  {
    mySerial.print("     ¡LIMITE EXCEDIDO!");
  }
  mySerial.println("");

  obd.read(PID_COOLANT_TEMP, value3);   //Lee temperatura del refrigerante (ºC)
  mySerial.print("Temperatura del refrigerante (ºC): ");
  mySerial.print(value3);
  if (value3 < 80)
  {
    mySerial.print("     COCHE FRIO");
  }
  if (value3 > 95)
  {
    mySerial.print("     ¡PELIGRO! ¡COCHE CALIENTE!");
  }
  mySerial.println("");

  obd.read(PID_ENGINE_LOAD, value4);    //Lee valor calculado de carga (%)
  mySerial.print("Valor calculado de carga (%): ");
  mySerial.println(value4);

  obd.read(0x14, value5);    //Lee voltaje del sensor de oxígeno (Short term fuel trim, Bank 1, Sensor 1)
```

```
  mySerial.print("Sensor de oxigeno 1 (%): ");
  mySerial.println(value5);

  obd.read(0x15, value6);    //Lee voltaje del sensor de oxígeno (Short term fuel trim, Bank 1, Sensor 2)
  mySerial.print("Sensor de oxigeno 2 (%): ");
  mySerial.println(value6);

  obd.read(PID_SHORT_TERM_FUEL_TRIM_1, value7);    //Lee "Short term fuel trim - Bank 1" (%)
  mySerial.print("Short term fuel trim (%): ");
  mySerial.print(value7);
  if (value7 < 0)
  {
   mySerial.print("     MEZCLA  ACTUAL RICA EN COMBUSTIBLE");
  }
  if (value7 > 0)
  {
   mySerial.print("     MEZCLA ACTUAL POBRE EN COMBUSTIBLE");
  }
  mySerial.println("");

  obd.read(PID_LONG_TERM_FUEL_TRIM_1, value8);   //Lee "Long term fuel trim - Bank 1" (%)
  mySerial.print("Long term fuel trim (%): ");
  mySerial.print(value8);
  if (value8 < 0)
  {
   mySerial.print("     MEZCLA MEDIA RICA EN COMBUSTIBLE");
  }
  if (value8 > 0)
  {
   mySerial.print("     MEZCLA MEDIA POBRE EN COMBUSTIBLE");
  }
  mySerial.println("");

  obd.read(PID_INTAKE_MAP, value9);   //Lee presión del colector de admisión de aire (kPa)
  mySerial.print("Presion del colector de admision de aire (kPa): ");
  mySerial.println(value9);

  obd.read(PID_INTAKE_TEMP, value10);   //Lee temperatura del aire de admisión (ºC)
  mySerial.print("Temperatura del aire de admision (ºC): ");
  mySerial.println(value10);

  obd.read(PID_TIMING_ADVANCE, value11);   //Lee avance de sincronización de chispa (º)
  mySerial.print("Avance de sincronizacion de chispa ( º): ");
  mySerial.println(value11);

  obd.read(PID_THROTTLE, value12);   //Lee posición del acelerador (%)
  mySerial.print("Posicion del acelerador (%): ");
  mySerial.println(value12);

  delay(2000);
```

```
 mySerial.print("\33[H\33[2J");     //Borra todo lo escrito antes de volver a escribir (Solo funciona en el programa Putty)
}
```

7.3.4. Programa definitivo de lectura y escritura de datos con Custodium

```
/* Código fuente de Custodium Tracker.  Calcula la posición GPS, lee datos de un vehículo, los analiza, genera las alertas necesarias y envía la información al servidor por http */

//Configurar debug
//#define DEBUG_ON
//#define DEBUG_RAM_ON
// Configurar usuario
#define CUSTODIUM
//Configurar acceso a internet
[…]
//Configurar timeouts
#define GPS_TIMEOUT 300000
//Configurar continuous mode (no sleep)
#define CONTINUOUS_MODE
//Tiempo de hibernación en minutos
#define SLEEP_PERIOD 5

//Se incluyen las librerias necesarias para el programa.
#include <SoftwareSerial.h>
[…]

//Librerías del cable OBD
#include <Wire.h>
#include "OBD.h"
COBD obd; /* for Model A (UART version) */

//Librería EEPROM
#include <EEPROM.h>

//Datos del RTC
[…]

//Variables que usa el cable OBD
int value1;
int value2;
int value3;
int value4;
```

```
int value5;

//Variables de tiempo y EEPROM
int years;
byte year1;
byte year2;
byte counter;

void setup()
{
 //Conexión serie.
 #ifdef DEBUG_ON
  Serial.begin(9600);
  delay(1000);
  Serial.println(F("Custodium Tracker 1.0"));
  Serial.println(F("DEBUG ON"));
  Serial.println(F("CUSTODIUM version"));
 #endif

 //Inicialización del RTC
 […]
}

void loop()
{
 //Creación del objeto GPS y variables
[…]

//Creación de objeto InetGSM y variables
[…]

 #ifdef DEBUG_ON
  Serial.println(F("Entering loop"));
 #endif

 //Alarma
  […]
  if (energy.WasSleeping()) {
   #ifdef DEBUG_ON
    Serial.println(F("  Interrupt and was sleeping"));
   #endif }
  else {
   #ifdef DEBUG_ON
    Serial.println(F("  Interrupt and was NOT sleeping"));
   #endif }
 }

 #ifdef CONTINUOUS_MODE
 while(1){
 #endif

 if (gsm.begin(2400)){
```

```
  #ifdef DEBUG_ON
    Serial.println(F("GSM READY"));
  #endif
  gsm.forceON();          //Para asegurarse de que la SIM908 funciona
  started=true;  }
 else{
  #ifdef DEBUG_ON
    Serial.println(F("GSM IDLE"));
  #endif }

 if(started){
  i=0;
  //Adjuntar GPRS
  […]
  #ifdef DEBUG_ON
    Serial.println(F("attachGPRS READY"));
  #endif
  delay(1000);

  #ifdef DEBUG_ON
    Serial.print(F("IP address:"));
    […]
    delay(5000);
  #endif

  //Adjuntar GPS
  If  […] {
    #ifdef DEBUG_ON
      Serial.println(F("GPS READY"));
    #endif }
  else {
    #ifdef DEBUG_ON
      Serial.println(F("GPS ERROR"));
    #endif }
 }

 //Obtener información del GPS
  […]

//Obtener lon, lat, alt, time y vel del GPS
  […]
  delay(1000);

  time[4]=0;   //Finaliza la cadena time en la quinta posición para quedarse con el año.
  years=atoi(time);    //Transforma la cadena de caracteres time en un entero.

//        // Comandos para iniciar a cero manualmente las funciones que usen años:
//        // Desactivadas hasta su uso:
//
//        //Neumáticos:
//        year1=years-2000;               //year1
//        EEPROM.write(0,year1);
```

```
//
//      //ITV. Para apagar la alarma de ITV pero seguir contando.
//      EEPROM.write(3,0);
//
//      //ITV. Para poner todo a cero (coche nuevo).
//      EEPROM.write(3,0);
//      year2=years-2000;               //year2
//      EEPROM.write(1,year2);
//      EEPROM.write(2,0);

       #ifdef DEBUG_ON
        Serial.println();
        Serial.println(F("GPS DATA:"));
        Serial.print(F("Latitude: "));
            Serial.println(lat);
        Serial.print(F("Longitude: "));
            Serial.println(lon);
        Serial.print(F("Altitude: "));
            Serial.println(alt);
        Serial.print(F("Time: "));
            Serial.println(time);
        Serial.print(F("Sats: "));
            Serial.println(num);
        Serial.print(F("Velocity: "));
            Serial.println(vel);
       #endif

      #ifdef DEBUG_ON
          if(stat==1)
                  Serial.println(F("NOT FIXED"));
          else if(stat==0)
                  Serial.println(F("UNKNOW"));
          else if(stat==2)
                  Serial.println(F("2D FIXED"));
          else if(stat==3)
                  Serial.println(F("3D FIXED"));
      #endif

       //Obtener información de la batería
        […]
       #ifdef DEBUG_ON
        Serial.println();
        Serial.println(F("BATTERY INFO"));
        if (a[0]=='0') Serial.println(F(" Not charging"));
        if (a[0]=='1') Serial.println(F(" Charging"));
        if (a[0]=='2') Serial.println(F(" Charging finished"));
        Serial.print(msg1);
        Serial.println(F("%"));
        Serial.print(msg2);
        Serial.println(F("mV"));
        Serial.print("Batt:");
        Serial.println(batt);
```

```
        #endif
        free(msg1);
        free(msg2);
        free(a);
        if (atoi(batt)<=BATT_ALARM) {
          alert=1;
          #ifdef DEBUG_ON
            Serial.println(F("LOW BATTERY ALARM"));
          #endif  }

        #ifdef DEBUG_RAM_ON
          print_ram();
        #endif

        //Obtener IMEI
         […]
        #ifdef DEBUG_ON
          Serial.println();
          Serial.print(F("IMEI: "));
               Serial.println(imei_tmp);
        #endif
         […]

        #ifdef   CUSTODIUM
         […]

//Iniciar el cable
obd.begin();   //Se inicia el cable OBD-II
while (!obd.init());   //Permanece en este paso hasta conexión OBD-II operativa.
elay(1000);

//Comandos de lectura del cable
obd.read(PID_RPM,value1);   //Lee revoluciones (rpm)
obd.read(PID_SPEED, value2);   //Lee velocidad (km/h)
obd.read(PID_COOLANT_TEMP, value3);   //Lee temperatura del refrigerante (ºC)

//Funciones para generar alertas.

//Alerta de revoluciones. Manda un aviso cuando las rpm superan 5000.
        if (value1 > 5000)
        {
          alert= (alert*10)+2;
        }

//Alerta de velocidad. Manda un aviso cuando la velocidad es mayor de 130km/h.
        if (value2 > 130)
        {
          alert= (alert*10)+3;
        }

//Alerta de temperatura de refrigerante. Avisa cuando la temperatura supera 95ºC.
        if (value3 > 95)
```

```
        {
          alert= (alert*10)+4;
        }

//Alerta de localización. Manda un aviso cuando la posición GPS se sale de unos márgenes.
        if (lat_float>41.683056000 || lon_float>-0.884484170 || lat_float<41.680355000 ||
        lon_float<-0.887610140 )
        {
          alert= (alert*10)+5;
        }

//Aviso de neumáticos. Manda un aviso cuando los neumáticos no se han cambiado en 5 años.
        if (tire_time()==1)
        {
          value4 = 1;
        }
        else
        {
          value4 = 0;
        }

//Aviso de ITV. Manda un aviso cuando el contador de ITV es VERDADERO.
        if (eeprom_itv()==1)
        {
          EEPROM.write(3,1);     //Guarda un valor de 1 en la EEPROM
        }
        value5 = EEPROM.read(3);

//Envía los datos al servidor web.
  […]
sprintf(msg,"%s?imei=%s&latitude=%s&longitude=%s&speed=%s&battery=%s&alert=%d&oth
er=Rev:%d,Vel:%d,Temp-Refr:%d,,,NEUMATICO:%d,,ITV:%d",
URL_SEND,imei_tmp,lat,lon,vel,batt,alert,value1,value2,value3,value4,value5);
        #ifdef DEBUG_ON
          Serial.println(msg);
        #endif

//free lon lat alt time y vel
        free(lon);
        free(lat);
        free(num);
        free(alt);
        free(time);
        free(vel);
        free(msg);
        free(batt);
        free(imei_tmp);
       […]

  alert=0;

  #ifdef DEBUG_RAM_ON
```

```
  print_ram();
 #endif

 #ifdef CONTINUOUS_MODE
 }
 #endif

 #ifdef DEBUG_ON
  Serial.println(F("Powering off SIM"));
 #endif

 […]
 delay(5000);

 //Hibernar
 #ifdef DEBUG_ON
  Serial.println(F("Setting next alarm"));
 #endif
 […]

 //Hibernar
 #ifdef DEBUG_ON
  Serial.println(F("Powering down"));
 #endif
 energy.PowerDown();
};    //Final de bucle

void set_next_alarm(void)
{
 […]
  #ifdef DEBUG_ON
   //t.year, t.mon, t.mday, t.hour, t.min, t.sec
   Serial.println(F("RTC date and time"));
   Serial.print(t.year);
   Serial.print(' ');
   Serial.print(t.mon);
   Serial.print(' ');
   Serial.print(t.mday);
   Serial.print(' ');
   Serial.print(t.hour);
   Serial.print(':');
   Serial.print(t.min);
   Serial.print(':');
   Serial.print(t.sec);
   Serial.println(' ');
  #endif
  […]

  // Configurar Alarm2.
  […]
  // Activar Alarm2
  […]
```

```
}

//ISR
void INT0_ISR(void)
{
  […]
}

int8_t convert2Degrees(char* input)
{
  […]
}

#ifdef DEBUG_RAM_ON
void print_ram()
 {
  Serial.print(F("Available RAM Memory: "));
  Serial.println(availableMemory());
}

// Número de bytes libres actualmente en la RAM
int availableMemory()
 {
  […]
}
#endif


//Funciones necesarias para calcular valores y alertas:

//Contador de neumáticos. Cuenta 5 años para la alarma de neumáticos.
  boolean tire_time ()
  {
    boolean tire = 0;
    year1 = EEPROM.read(0);
    if (years-2000 > year1+4)
    {
      tire = 1;    //Activa la alarma hasta que los años se actualicen manualmente
    }
    return tire;
  }

//Contador de ITVs. Es VERDADERO cuando no ha pasado la ITV en: 4 años para la primera vez,
2 años hasta los 10 años, 1 año para después.
  boolean eeprom_itv ()
  {
    boolean itv = 0;
    year2 = EEPROM.read(1);
    counter = EEPROM.read(2);

    if (years-2000==year2+4 && counter==0)        //Primera ITV.
    {
```

```
   year2=years-2000;
   EEPROM.write(1,year2);     //Actualiza el año
   counter = 1;
   EEPROM.write(2,counter);     //Actualiza el contador
   itv = 1;
  }

  if (years-2000==year2+2 && (counter==1 || counter==2 || counter==3))  //Hasta los 10 años
  {
   year2=years-2000;
   EEPROM.write(1,year2);    //Actualiza el año
   counter++;
   EEPROM.write(2,counter);     //Actualiza el contador
   itv = 1;
  }

  if (years-2000==year2+1 && counter==4)       //El resto de ITVs.
  {
   year2=years-2000;
   EEPROM.write(1,year2);       //Actualiza el año
   itv = 1;
  }
 return itv;
}
```

7.3.5. Programa ideal de lectura y escritura de datos con Custodium

```
/* Código fuente de Custodium Tracker.  Calcula la posición GPS, lee datos de un vehículo
(algunos sin éxito), los analiza, genera las alertas necesarias y envía la información al servidor
por http */

//Configurar debug
//#define DEBUG_ON
//#define DEBUG_RAM_ON
// Configurar usuario
#define CUSTODIUM
//Configurar acceso a internet
[…]
//Configurar timeouts
#define GPS_TIMEOUT 300000
//Configurar continuous mode (no sleep)
#define CONTINUOUS_MODE
//Tiempo de hibernación en minutos
#define SLEEP_PERIOD 5
```

```
//Se incluyen las librerías necesarias para el programa.
#include <SoftwareSerial.h>
[…]

//Librerías del cable OBD
#include <Wire.h>
#include "OBD.h"
COBD obd; /* for Model A (UART version) */

//Librería EEPROM
#include <EEPROM.h>

//Datos del RTC
[…]

//Variables que usa el cable OBD
int value1;
int value2;
int value3;
int value4;
int value5;
int value6;
int value7;
int value8;
int value9;

//Variables de contadores de combustible y km
int fuel1;
byte fuel2;
byte count;
long fill;
byte fill2;
byte fill3;
long kmetres;
byte kmetres2;
byte kmetres3;

//Variables de tiempo
int years;
byte year1;
byte year2;
byte counter;
int itv_alert;

void setup()
{
  //Conexión serie.
  #ifdef DEBUG_ON
    Serial.begin(9600);
    delay(1000);
    Serial.println(F("Custodium Tracker 1.0"));
```

```
  Serial.println(F("DEBUG ON"));
  Serial.println(F("CUSTODIUM version"));
 #endif

 //Inicialización del RTC
 […]
}

void loop()
{
 //Creación del objeto GPS y variables
[…]

//Creación de objeto InetGSM y variables
[…]

 #ifdef DEBUG_ON
  Serial.println(F("Entering loop"));
 #endif

 //Alarma
  […]
  if (energy.WasSleeping()) {
   #ifdef DEBUG_ON
    Serial.println(F("  Interrupt and was sleeping"));
   #endif }
  else {
   #ifdef DEBUG_ON
    Serial.println(F("  Interrupt and was NOT sleeping"));
   #endif }
 }

 #ifdef CONTINUOUS_MODE
 while(1){
 #endif

 if (gsm.begin(2400)){
  #ifdef DEBUG_ON
   Serial.println(F("GSM READY"));
  #endif
  gsm.forceON();          //Para asegurarse de que la SIM908 funciona
  started=true;  }
 else{
  #ifdef DEBUG_ON
   Serial.println(F("GSM IDLE"));
  #endif }

 if(started){
  i=0;
  //Adjuntar GPRS
  […]
  #ifdef DEBUG_ON
```

```
    Serial.println(F("attachGPRS READY"));
   #endif
   delay(1000);

   #ifdef DEBUG_ON
    Serial.print(F("IP address:"));
    […]
    delay(5000);
   #endif

   //Adjuntar GPS
   If  […] {
    #ifdef DEBUG_ON
     Serial.println(F("GPS READY"));
    #endif }
   else {
    #ifdef DEBUG_ON
     Serial.println(F("GPS ERROR"));
    #endif }
  }

  //Obtener información del GPS
   […]

 //Obtener lon, lat, alt, time y vel del GPS
   […]
   delay(1000);

   time[4]=0;   //Finaliza la cadena time en la quinta posición para quedarse con el año.
   years=atoi(time);    //Transforma la cadena de caracteres time en un entero.

//        //Comandos para iniciar a cero manualmente las funciones que usen años:
//        //Desactivadas hasta su uso:
//
//        //Contador de gasolina y de recargas.
//        EEPROM.write(3,0);              //fill
//        EEPROM.write(4,0);              //count
//
//        //Neumáticos. Contador de km + contador de años.
//        EEPROM.write(5,0);              //kmetres
//        year1=years-2000;               //year1
//        EEPROM.write(0,year1);
//
//        //ITV. Para apagar la alarma de ITV pero seguir contando.
//        EEPROM.write(9,0);              //Alerta ITV
//
//        //ITV. Para poner todo a cero (coche nuevo).
//        EEPROM.write(9,0);              //Alerta ITV
//        year2=years-2000;               //year2
//        EEPROM.write(1,year2);
//        EEPROM.write(6,0);              //Counter
```

```
 #ifdef DEBUG_ON
  Serial.println();
  Serial.println(F("GPS DATA:"));
  Serial.print(F("Latitude: "));
       Serial.println(lat);
  Serial.print(F("Longitude: "));
       Serial.println(lon);
  Serial.print(F("Altitude: "));
       Serial.println(alt);
  Serial.print(F("Time: "));
       Serial.println(time);
  Serial.print(F("Sats: "));
       Serial.println(num);
  Serial.print(F("Velocity: "));
       Serial.println(vel);
 #endif

#ifdef DEBUG_ON
    if(stat==1)
              Serial.println(F("NOT FIXED"));
    else if(stat==0)
              Serial.println(F("UNKNOW"));
    else if(stat==2)
              Serial.println(F("2D FIXED"));
    else if(stat==3)
              Serial.println(F("3D FIXED"));
#endif

 //Obtener información de la batería
  [...]
 #ifdef DEBUG_ON
  Serial.println();
  Serial.println(F("BATTERY INFO"));
  if (a[0]=='0') Serial.println(F(" Not charging"));
  if (a[0]=='1') Serial.println(F(" Charging"));
  if (a[0]=='2') Serial.println(F(" Charging finished"));
  Serial.print(msg1);
  Serial.println(F("%"));
  Serial.print(msg2);
  Serial.println(F("mV"));
  Serial.print("Batt:");
  Serial.println(batt);
 #endif
 free(msg1);
 free(msg2);
 free(a);
 if (atoi(batt)<=BATT_ALARM) {
  alert=1;
  #ifdef DEBUG_ON
   Serial.println(F("LOW BATTERY ALARM"));
  #endif  }
```

```
        #ifdef DEBUG_RAM_ON
          print_ram();
        #endif

        //Obtener IMEI
         […]
        #ifdef DEBUG_ON
          Serial.println();
          Serial.print(F("IMEI: "));
              Serial.println(imei_tmp);
        #endif
         […]

        #ifdef   CUSTODIUM
         […]

//Iniciar el cable
obd.begin();  //Se inicia el cable OBD-II
while (!obd.init());  //Permanece en este paso hasta conexión OBD-II operativa.
elay(1000);

//Comandos de lectura del cable
obd.read(PID_RUNTIME, value1);  //Tiempo 1 (sec)
obd.read(PID_RPM, value2);  //Revoluciones (rpm)
obd.read(PID_SPEED, value3);  //Velocidad (km/h)
obd.read(PID_DISTANCE, value4);  //Distancia (km)
obd.read(PID_FUEL_LEVEL, value5);  //Nivel de combustible (%)
obd.read(PID_AMBIENT_TEMP, value6);  //Temperatura ambiente (ºC)
obd.read(PID_ENGINE_OIL_TEMP, value7);  //Temperatura del aceite (ºC)
obd.read(PID_COOLANT_TEMP, value8);  //Temperatura del refrigerante (ºC)
obd.read(0x40, value9);  //Número de averías en el vehículo

//Funciones para generar alertas.

//Cantidad de combustible rellenado (%) y contador de recargas. Muestra el número de veces
que el depósito se llena y la cantidad que se llena.
 fuel2=EEPROM.read(2);
 fuel1=fuel2;
 fuel2=value5;
 EEPROM.write(2,fuel2);       //Actualiza en la EEPROM el nivel de fuel en el depósito.
 if (fuel2 > fuel1)        //Depósito repostado
 {
  fill2=EEPROM.read(3);       //Contador de % inferior a 250
  fill3=EEPROM.read(8);       //Contador de bloques de 250
  fill = 250*fill3 + fill2;       //Cálculo del % total. Para guardar en la EEPROM, max 255. Hay que
llevar dos contadores a la vez.
  fill=fill+(fuel2-fuel1);        //Como el depósito se ha repostado, se actualiza el valor total de
fuel repostado.
  fill2 = fill - 250*fill3;
  if (fill2 > 250)         //Si la variable inferior a 250 llega a 250, se pone a cero y se aumenta en
uno el otro contador.
  {
```

```
    fill2 = fill2 - 250;
    fill3++;
    EEPROM.write(8,fill3);
   }
   EEPROM.write(3,fill2);
   count=EEPROM.read(4);
   count++;
   EEPROM.write(4,count);          //Como el depósito se ha repostado, se aumenta en uno el
número total de repostajes.
  }

//Alarma de fallo.
  if (value9 != 0)
  {
   alert= (alert*10)+2;
  }

//Alarma de velocidad. Manda un aviso cuando la velocidad es mayor de 130km/h.
  if (value3 > 130)
  {
   alert= (alert*10)+3;
  }

//Alarma de revoluciones. Manda un aviso cuando las rpm superan 5000.
  if (value2 > 5000)
  {
   alert= (alert*10)+4;
  }

//Alarma de tiempo transcurrido. Manda un aviso cuando el motor no se ha parado en 2 horas.
  if (value1 > 7200)
  {
   alert= (alert*10)+5;
  }

//Alarma de temperatura de aceite. Manda un aviso cuando la temperatura del aceite supera
120ºC.
  if (value7 > 120)
  {
   alert= (alert*10)+6;
  }

//Alarma de temperatura de refrigerante. Manda un aviso cuando la temperatura del
refrigerante supera 100ºC.
  if (value8 > 100)
  {
   alert= (alert*10)+7;
  }

//Alarma de neumáticos. Manda un aviso cuando los neumáticos no se han cambiado en
40000km o 5 años.
  if (km_count() > 40000 || tire_time()==1)
```

```
 {
  alert= (alert*10)+8;
 }

//Alarma de ITV. Manda un aviso cuando el contador de ITV es VERDADERO.
 if (eeprom_itv()==1)
 {
  EEPROM.write(9,1);     //Guarda un valor de 1 en la EEPROM
 }
 itv_alert = EEPROM.read(9);
 if  (itv_alert==1)              //Variable secundaria que se repite hasta que se pone a cero manualmente, ya que la primera variable solo se activa durante un ciclo para seguir contando.
 {
  alert= (alert*10)+9;
 }

//Alerta de localización. Manda un aviso cuando la posición GPS se sale de unos márgenes.
 if  (lat_float>41.683056000||lon_float>-0.884484170||lat_float<41.680355000||  lon_float<
 -0.887610140 )
   {
    alert= (alert*10)+10;
   }

//Envía los datos al servidor web.
 […]
sprintf(msg,"%s?imei=%s&latitude=%s&longitude=%s&speed=%s&battery=%s&alert=%lu&oth
er=running-time:%d;,rpm:%d;,speed:%d;,distance:%lu;,fuel-level:%d,fuel-charges:%d,
fuel-filled:%lu;,ambient-temp:%d;,oil-temp:%d;,coolant-temp:%d",
URL_SEND,imei_tmp,lat,lon,vel,batt,alert,value1,value2,value3,value4,value5,count,fill,value6,
value7,value8);

#ifdef DEBUG_ON
 Serial.println(msg);
#endif

//free lon lat alt time y vel
        free(lon);
        free(lat);
        free(num);
        free(alt);
        free(time);
        free(vel);
        free(msg);
        free(batt);
        free(imei_tmp);
       […]

 alert=0;

 #ifdef DEBUG_RAM_ON
  print_ram();
 #endif
```

```
  #ifdef CONTINUOUS_MODE
  }
  #endif

  #ifdef DEBUG_ON
    Serial.println(F("Powering off SIM"));
  #endif

  […]
  delay(5000);

  //Hibernar
  #ifdef DEBUG_ON
    Serial.println(F("Setting next alarm"));
  #endif
  […]

  //Hibernar
  #ifdef DEBUG_ON
    Serial.println(F("Powering down"));
  #endif
  energy.PowerDown();
};     //Final de bucle

void set_next_alarm(void)
{
  […]
    #ifdef DEBUG_ON
      //t.year, t.mon, t.mday, t.hour, t.min, t.sec
      Serial.println(F("RTC date and time"));
      Serial.print(t.year);
      Serial.print(' ');
      Serial.print(t.mon);
      Serial.print(' ');
      Serial.print(t.mday);
      Serial.print(' ');
      Serial.print(t.hour);
      Serial.print(':');
      Serial.print(t.min);
      Serial.print(':');
      Serial.print(t.sec);
      Serial.println(' ');
    #endif
    […]

    // Configurar Alarm2.
    […]
    // Activar Alarm2
    […]
}
```

```
//ISR
void INT0_ISR(void)
{
  […]
}

int8_t convert2Degrees(char* input)
{
  […]
}

#ifdef DEBUG_RAM_ON
void print_ram()
 {
  Serial.print(F("Available RAM Memory: "));
  Serial.println(availableMemory());
}

// Número de bytes libres actualmente en la RAM
int availableMemory()
 {
  […]
}
#endif


//Funciones necesarias para calcular valores y alertas:

//Contador de neumáticos. Cuenta 5 años para la alarma de neumáticos.
  boolean tire_time ()
  {
    boolean tire = 0;
    year1 = EEPROM.read(0);
    if (years-2000 > year1+4)
    {
      tire = 1;     //Activa la alarma hasta que los años se actualizan manualmente.
    }
    return tire;
  }

//Contador de km. Memoriza los km si se corta la corriente.
  int km_count ()
  {
    kmetres2=EEPROM.read(5);      //Contador de km inferior a 250
    kmetres3=EEPROM.read(7);      //Contador de bloques de 250
    kmetres = 250*kmetres3 + kmetres2;          //Cálculo de los km totales. Para guardar en la
EEPROM, max 255. Hay que llevar dos contadores a la vez.

    if (value4 < kmetres)       //Se ha cortado la corriente y value4 ha vuelto a empezar.
    {
      kmetres = kmetres + value4;        //Actualiza los km.
      kmetres2 = kmetres - 250*kmetres3;
```

```
    if (kmetres2 > 250)              //Si la variable inferior a 250 llega a 250, se pone a cero y se aumenta en uno el otro contador.
    {
      kmetres2 = kmetres2 - 250;
      kmetres3++;
      EEPROM.write(7,kmetres3);
    }
   }
   else      //No se ha cortado la corriente.
   {
    kmetres = value4;
    kmetres2 = kmetres - 250*kmetres3;
    if (kmetres2 > 250)              //Si la variable inferior a 250 llega a 250, se pone cero y se aumenta en uno el otro contador.
    {
      kmetres2 = kmetres2 - 250;
      kmetres3++;
      EEPROM.write(7,kmetres3);
    }
   }
   EEPROM.write(5,kmetres2);
   return kmetres;
 }

//Contador de ITVs. Es VERDADERO cuando no ha pasado la ITV en: 4 años para la primera vez, 2 años hasta los 10 años, 1 año para después.
 boolean eeprom_itv ()
 {
   boolean itv = 0;
   year2 = EEPROM.read(1);
   counter = EEPROM.read(6);

   if (years-2000==year2+4 && counter==0)       //Primera ITV.
   {
    year2=years-2000;
    EEPROM.write(1,year2);     //Actualiza el año
    counter = 1;
    EEPROM.write(6,counter);     //Actualiza el contador
    itv = 1;
   }

   if (years-2000==year2+2 && (counter==1 || counter==2 || counter==3))     //Siguientes ITVs hasta los 10 años.
   {
    year2=years-2000;
    EEPROM.write(1,year2);    //Actualiza el año
    counter++;
    EEPROM.write(6,counter);     //Actualiza el contador
    itv = 1;
   }

   if (years-2000==year2+1 && counter==4)       //El resto de ITVs.
```

```
  {
    year2=years-2000;
    EEPROM.write(1,year2);        //Actualiza el año
    itv = 1;
  }
 return itv;
}
```

7.4. ANEXO 4: ANÁLISIS DEL MERCADO

Para conocer el éxito y la rentabilidad del producto diseñado, lo primero que hay que hacer es identificar los posibles competidores y aquellos sistemas que realizan tareas muy similares a las que realiza este, con el fin de saber cómo diferenciar mi producto de otros ya existentes y descubrir cómo hacerse un posible hueco en el mercado.

Lo primero es fijarse en los productos más populares y más vendidos en los últimos años relacionados con los comandos OBD. Estos instrumentos ya han sido nombrados en el apartado de herramientas, pero debido a su éxito, es necesario volver a hablar de ellos para explicar las ventajas de este proyecto respecto a ellos.

Se trata de herramientas económicas y fáciles de usar que permiten conocer información de los vehículos, como es el caso de kits de diagnóstico con la máquina y el cable necesario, adaptadores con puerto USB y conectores con bluetooth que permiten la lectura de datos desde un teléfono móvil utilizando una aplicación disponible.

Figura 51. Kit de diagnosis OBD.

Figura 52. Cable OBD con USB.

Figura 53. Adaptador Bluetooth.

Atualmente existen múltiples empresas dedicadas a la venta de estos productos, como es el caso de “OBDCenter”, “Cocheobd2”, “Innova” o “Diagnosisauto”, aunque por supuesto, también se pueden encontrar en “Amazon” o “Google Shopping”.

Pero las ventajas de mi sistema respecto a estos más simples, son varias. En primer lugar, la limitada longitud de sus cables permite únicamente consultar datos dentro del vehículo, incluso utilizando el adaptador con bluetooth, ya que su distancia máxima de funcionamiento es de 10 metros y no permite alejarse mucho. Mientras que el sistema descrito en este proyecto, envia datos de manera automática a un servidor de internet que permite su consulta desde cualquier lugar del mundo. Además, este servidor puede recoger la información de varios vehículos simultaneamente, mientras que con otros adaptadores solo se puede extraer información del vehículo al que se conecte el cable o el adaptador bluetooth.

Pero la ventaja más importante es, que mi sistema permite trabajar con la información obtenida, pasando los datos leídos a una placa capaz de realizar cálculos, generar alertas y enviar los resultados. Algo que las otras herramientas no pueden hacer, ya que solo muestran los valores leídos.

Otro competidor, relacionado con la lectura de comandos pero en este caso también con la localización GPS, es “IMETRIK”, una empresa canadiense que ofrece la instalación de dispositivos en múltiples vehículos, que envían datos de posición y conducción, a través de una red móvil, a un servidor de internet o a una aplicación de teléfono, proporcionada por la empresa o elegida por el cliente. A pesar de la poca información que ofrece su página web, se puede interpretar que se basan en el mercado de los seguros de accidentes. Su sistema analiza principalmente parámetros relacionados con los movimientos del vehículo, como pueden ser la velocidad, la distancia recorrida, la localización GPS o los horarios, variables que son mostradas al cliente, en este caso una empresa de seguros, para saber según esta información, si deben hacerse responsables del incidente y si pueden ofrecer un precio más atractivo a usuarios que supongan un riesgo menor o que usen menos su vehículo. También permite la localización en caso de robo.

Se trata de un sistema parecido al desarrollado en este proyecto, pero la principal diferencia es, que IMETRIK no se centra en las averías ni en avisos para evitar posibles riesgos, sino en el uso que se hace de los vehículos. Esto, junto con la ventaja de que mi sistema admite sensores extra y resulta útil para más sectores como el de gestión de flotas, creación de rutas, estudios de mercado o alquiler de coches, proporciona una herramienta con más posibilidades que la diseñada por esta empresa.

Otra empresa que haría competencia a este proyecto, es la empresa “MapmyIndia”, la cual proporciona un sistema que se encarga de la localización GPS, la lectura de parámetros de los vehículos y el envío de la información a un dispositivo móvil, con el

objetivo de gestionar flotas y permitir el ahorro de grandes cantidades de dinero. Esto se consigue, según su página web, eligiendo rutas más eficientes en términos de combustible, probablemente después de analizar el consumo de varios vehículos, pero también rastreando el recorrido, controlando el tiempo y contando las paradas de los trayectos más frecuentes para reducir horas extra. Además ofrecen prevención de robos y la posibilidad de recibir alertas o alarmas cuando algo no funcione como debería.

También habría que considerar la empresa "Cartrack", que dispone de múltiples franquicias y entidades por todo el mundo, especialmente en África, y que ofrece un sistema que muestra a distancia la información necesaria. Según ellos, no disponen de ningún tipo de software en el sistema, ya que envían la información directamente a la web, donde es tratada. Proporcionan posicionamiento GPS, monitorización del comportamiento del vehículo, alertas en tiempo real e incluso sistemas para recuperación de vehículos robados. Como la mayoría de empresas utilizando esta tecnología, se centran en compañías de seguros, permitiéndoles calcular los riesgos y minimizar costes utilizando información sobre el comportamiento de sus clientes al volante.

Por último, otro posible competidor sería la empresa china "Sinocastel", la cual combina todas las funciones anteriores, ofreciendo un sistema útil para control de flotas o empresas aseguradoras, pero también para vehículos individuales y familiares. Permite visualizar el comportamiento y la localización de varios vehículos desde una página web o una aplicación móvil, ofrece el hardware necesario para compañías de seguros y también permite a las familias conocer el funcionamiento de su coche o localizar a sus hijos y mascotas.

Como se puede observar, estas tres últimas empresas sí que ofrecen un servicio muy similar al que proporcionaría mi sistema y suponen por tanto una competencia directa. De manera que, para diferenciar mi producto del suyo, serían necesarias algunas de las ampliaciones propuestas en el *Anexo 5*.

Evidentemente, existen más empresas relacionadas con el tema que las aquí mostradas, pero parecen ser pequeñas y no tener mucha repercusión en el mercado, como se deduce de la escasa información que muestran en su página o de los pocos servicios que ofrecen. Muchas de ellas se centran únicamente en la localización GPS.

En resumen, después de mucho investigar y dejando a un lado los simples productos OBD que solo permiten la lectura de datos, las empresas más grandes encontradas que realizan funciones parecidas a las de este proyecto son, la empresa IMETRIK, con sede en Montreal, la empresa MapmyIndia, con sede en Nueva Delhi, la empresa Cartrack, con sede mayoritariamente en África, y la empresa Sinocastel, con sede en Shenzhen,

lo cual se traduce en una reducida competencia internacional y un posible monopolio en el mercado español.

7.5. ANEXO 5: POSIBLE AMPLIACIÓN

Debido a las limitaciones surgidas en la realización del proyecto por la escasez de herramientas y por tratarse de un sistema pionero con poca información disponible para consultar, este trabajo no abarca todas sus posibilidades ni resulta tan útil en algunos aspectos como cabía esperar. Aun así, supone un buen punto de partida para futuras ampliaciones con mayor presupuesto, herramientas más modernas y nuevas investigaciones.

Un ejemplo de esto, se puede ver fácilmente en "CUSTODIUM Soluciones y Sistemas S.L.", la empresa en la que he realizado este proyecto. Cuando empecé con él, la única forma existente de conectar Custodium Tracker con un ordenador era utilizando un FTDI, lo cual resultaba algo complejo y requería varios cables. Pero como no existía otra manera, así se ha realizado en el proyecto. Actualmente, una vez acabado dicho proyecto, la empresa ha terminado el desarrollo de una nueva placa de expansión, que además de los pines y conexiones de modelos anteriores, también contiene un puerto Mini USB. Esta placa se adapta perfectamente a Custodium Tracker y permite la conexión con un ordenador sin necesidad de usar un FTDI, lo cual reduce el tamaño del sistema, el número de cables usados y las conexiones, como se puede ver en la siguiente imagen. Esta mejora está disponible para futuras ampliaciones o ensayos y supone una ventaja considerable.

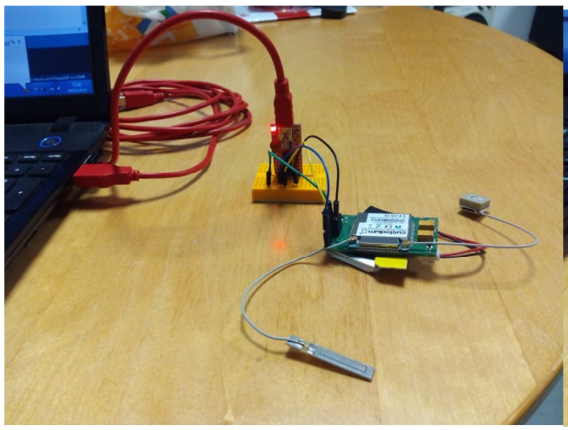


Figura 54. Antigua conexión Custodium – Ordenador.

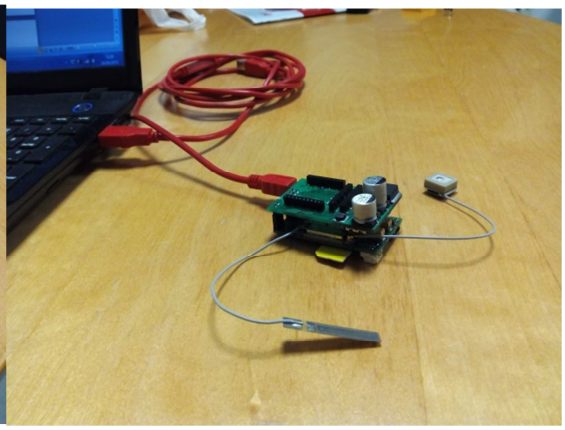


Figura 55. Nueva conexión Custodium – Ordenador.

Respecto al resto de problemas aún sin solucionar que han surgido en la realización del proyecto, el primero es el escaso número de datos que pueden ser leídos. Esto podría ser debido al cable utilizado o al modelo de vehículo usado.

El cable empleado en este proyecto es el llamado OBD-II UART ADAPTER, modelo A, ya que es el único del que disponía y he podido utilizar. Como ya se ha dicho anteriormente, los parámetros disponibles para lectura son elección del fabricante del vehículo, pero dentro de estos parámetros elegidos, existe la posibilidad de que unos cables lean más datos que otros. Una posible ampliación de este proyecto sería por tanto, repetir todo el proceso usando el código diseñado pero utilizando otros cables más sofisticados y caros para la lectura de valores.

Otra explicación podría ser el modelo de vehículo utilizado. Como ya se ha explicado, he probado el sistema en tres vehículos diferentes sin observar grandes cambios entre ellos, pero es posible que existan otros que permitan la lectura de un mayor número de datos. De nuevo tengo la limitación de no poder ensayar en más coches, pero este proyecto se podría ampliar probando el sistema aquí diseñado en un mayor número de vehículos, incluyendo camiones y autobuses, para poder sacar conclusiones de las diferencias observadas o encontrar un modelo que funcione mejor y aumente la utilidad del sistema.

Se cree que con el paso de los años, más parámetros serán mostrados debido a la aparición de nuevas herramientas o a la concesión de los fabricantes por petición de los usuarios. En caso contrario, otra opción podría ser que las empresas interesadas en un sistema como este, hablen con alguna empresa de vehículos y pacten con el fabricante los parámetros que se permiten leer, de manera que se diseñen ejemplares distintos para un pedido concreto de coches.

Adicionalmente, se podrían incorporar al vehículo sensores extra que se comunicasen con Custodium Tracker y le dieran al sistema información añadida para enviar al servidor. Podrían conectarse con la placa mediante cables o de manera inalámbrica, incorporando un módulo apropiado. Ambas opciones necesitarían un cambio en las conexiones de Custodium y aumentarían el tamaño del sistema, pero es posible y sería rentable teniendo en cuenta las ventajas. Esto aumentaría la funcionalidad del sistema y reduciría el problema de la falta de comandos mostrados.

Otro problema encontrado, de poca importancia y con una posible solución, es el tiempo de lectura de datos. Éstos son leídos en tiempo real, pero debido a la poca velocidad del cable y al tiempo necesario para detectar la posición GPS, los datos son enviados cada cierto número de segundos, lo cual no proporciona una visión exacta de lo que sucede en el vehículo. Esto es preferible, ya que de esta manera se envían menos datos al servidor y la información se ve con más claridad, pero la precisión es menor.

En el caso de querer aumentarla para observar los datos instantáneos y ver la tendencia que siguen, con el objetivo de poder detectar errores más fácilmente, se podría utilizar un programa informático especializado en datos OBD que permite su

visualización mediante gráficas, como se verá más adelante. El problema es que esto tendría que realizarse "in situ", usando el ordenador dentro del vehículo, lo cual eliminaría la ventaja de ver la información a distancia.

También me topé con el problema de la detección de averías, pero no disponía de ningún coche averiado, así que no pude ensayar o investigar mucho sobre el tema. Existe un comando OBD (ver *Anexo 1*) que indica el número de errores en el vehículo, pero igual que muchos otros, al leerlo con el cable su valor es cero, y en este caso no puedo saber si es por ausencia de fallos en el vehículo o porque el cable no lee dicho parámetro. Por tanto, este proyecto se podría ampliar profundizando en el tema y ensayando con otros cables en vehículos averiados.

De todas formas, el sistema diseñado únicamente mandaría información sobre la existencia de averías o el número de ellas, pero no explicaría de qué problema se trata, ya que las posibilidades pueden ser muy variadas y su explicación muy extensa para enviarla al servidor. Por tanto, una vez confirmada la existencia de uno o varios fallos en el vehículo, sería necesario conectar "in situ" una máquina de diagnóstico o un ordenador con un programa especializado de averías. Este último resulta mucho más económico debido a la variedad existente de ellos actualmente y su reducido precio, algunos incluso gratuitos.

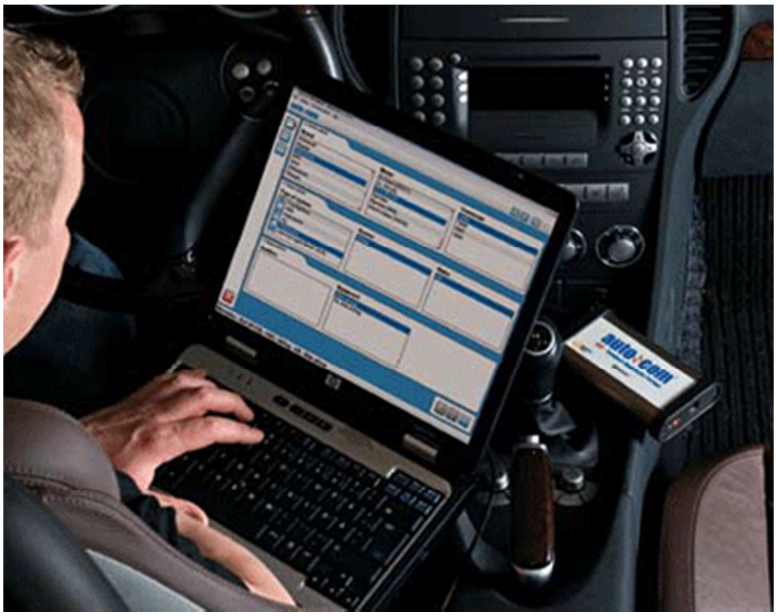


Figura 56. Diagnóstico de un vehículo con un programa informático especializado.

Tal es el caso de programas como "HealTech" o "Auterra Dyno-Scan", que permiten conocer el error exacto existente en el vehículo y su posible causa, así como eliminarlo del registro del automóvil una vez solucionado. Existe información más detallada sobre

ellos en la bibliografía, junto con la lista completa de posibles averías con su código correspondiente y el sistema de codificación usado.

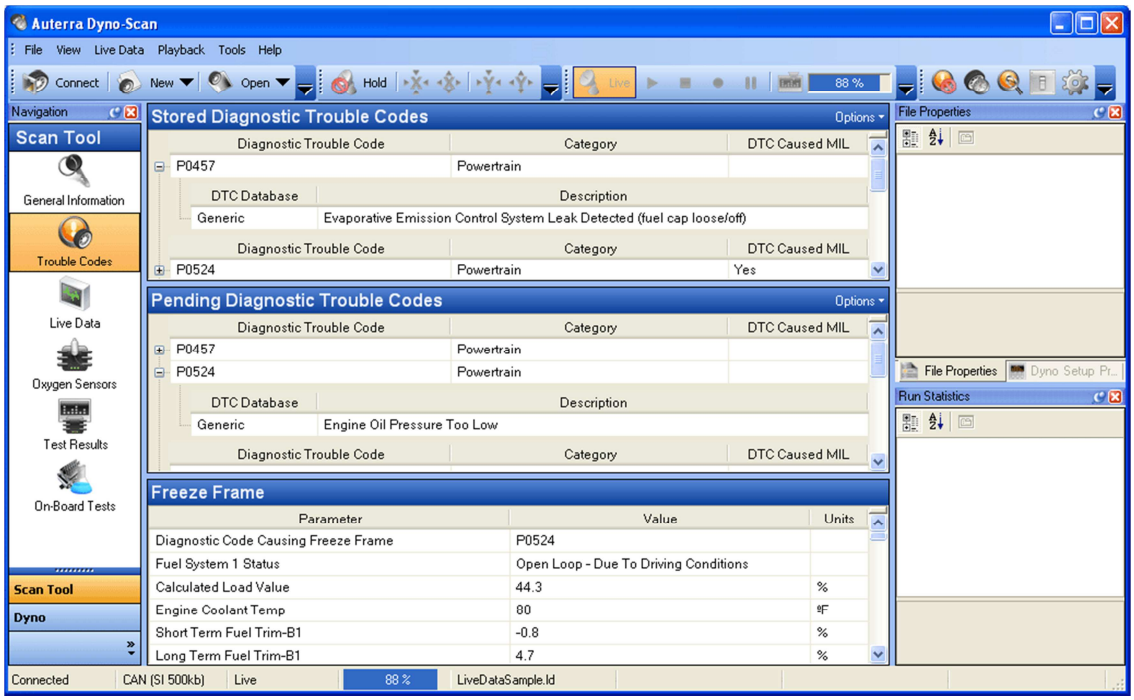


Figura 57. Auterra Dyno-Scan. Información de averías.

Estos mismos programas también permiten visualizar los datos mediante gráficas, lo cual, como ya se ha explicado, resulta muy útil para aumentar la precisión de los datos observados y no perder información debido a una lectura lenta.

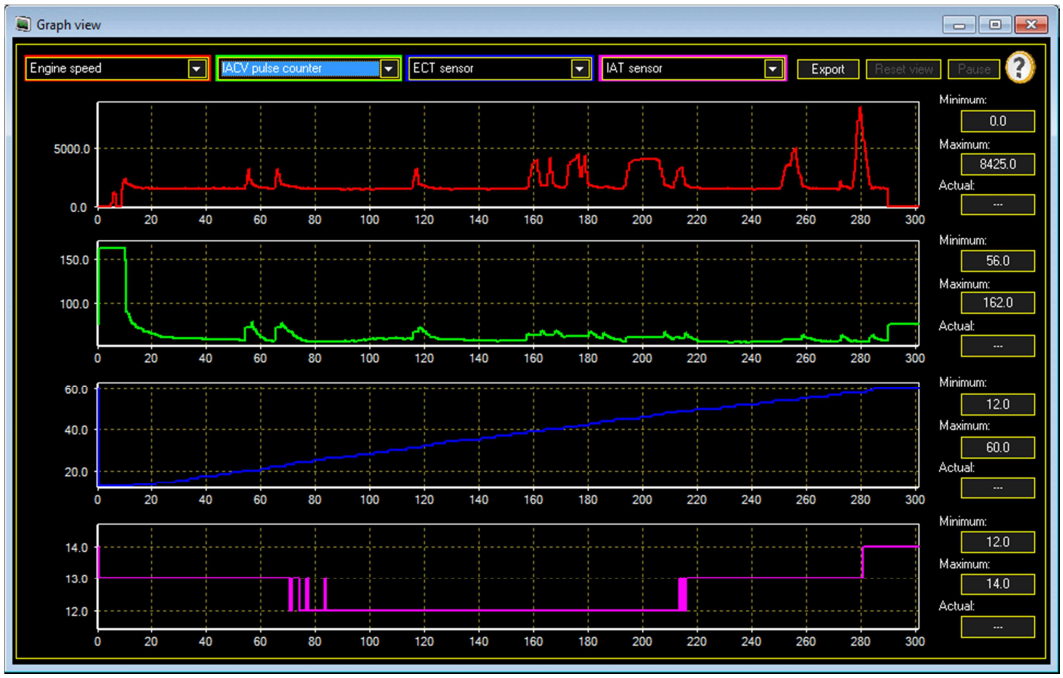


Figura 58. Healtech. Gráficas.

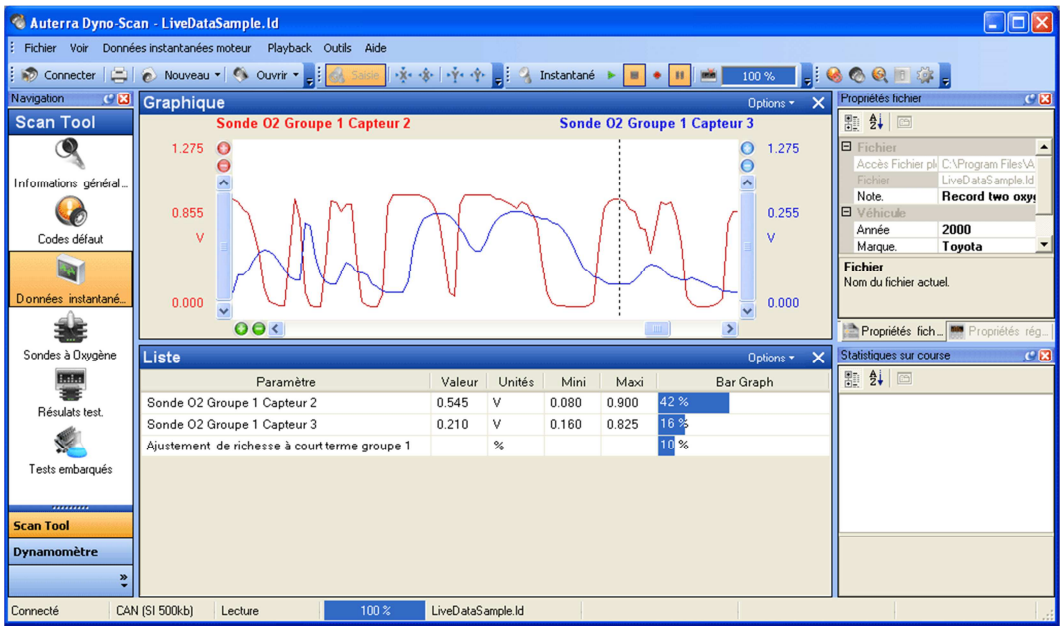


Figura 59. Auterra Dyno-Scan. Gráficas.

Otra opción sería utilizar diversas aplicaciones móviles como “Torque” para realizar estas funciones, pero para ello habría que usar un lector OBD con bluetooth, que al igual que la conexión con ordenador, tiene un alcance reducido de pocos metros, y además impide disponer del resto de ventajas descritas en este proyecto. Por tanto, es preferible utilizar el ordenador, ya que además de disponer de programas más sofisticados con muchas más opciones, no obliga a cambiar el lector, lo que permite seguir usando el sistema y almacenar información en internet a la vez que se visualiza con estos programas.

Por último, una de las ampliaciones más complicadas pero a la vez una de las más útiles, sería la actuación sobre el propio vehículo. Hasta el momento solo se había hablado de recibir información de la centralita, pero también podría resultar muy ventajoso enviarle información para manipular su funcionamiento. Esto podría permitir por ejemplo, detener el vehículo desde un teléfono móvil en caso de robo o modificar parámetros como la inyección de gasolina, con el fin de obtener más potencia o reducir el consumo, según lo que se desee.

Estos ejemplos son bastante factibles, ya que alguna de las empresas citadas en el *Anexo 4* ya trabajan en ello, pero resulta difícil y caro investigar sobre el tema debido al riesgo que supone para los vehículos en los que se ensaye, ya que manipular erróneamente una centralita puede inutilizarla.

8. BIBLIOGRAFÍA

Historia:

e-auto. (s.f.). ¿Qué es OBD-II?. Recuperado de http://e-auto.com.mx/manual_detalle.php?manual_id=119

On-board diagnostics, (s.f.). En Wikipedia. Recuperado de http://en.wikipedia.org/wiki/On-board_diagnostics#History

OBD-II PIDs, (s.f.). En Wikipedia. Recuperado de http://en.wikipedia.org/wiki/OBD-II_PIDs

Herramientas:

Conector:

Aficionados a la Mecánica. (s.f.). OBD (ON BOARD DIAGNOSTIC). Recuperado de http://www.aficionadosalamecanica.com/obd2.htm

e-auto. (s.f.). ¿Qué es OBD-II?. Recuperado de http://e-auto.com.mx/manual_detalle.php?manual_id=119

Cable:

ArduinoDev. (s.f.). OBD-II Adapter for Arduino. Recuperado de http://arduinodev.com/hardware/obd-kit/

Arduino:

Arduino. (s.f.). *Sin título*. Recuperado de http://www.arduino.cc/es/pmwiki.php?n

Arduino. (s.f.). Arduino UNO. Recuperado de http://arduino.cc/en/Main/ArduinoBoardUno

Custodium tracker:

Custodium tracker. (s.f.). Qué es Custodium. Recuperado de http://custodiumtracker.com/que-es-custodium/

FTDI:

FTDI, (s.f.). En Wikipedia. Recuperado de http://es.wikipedia.org/wiki/FTDI

Proceso:

Vídeos:

[BuckoHD]. (2014, Marzo 27). Bluetooth OBD II Tutorial Super Mini ELM327 [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=WOQLh1iGN60

[Carscan7]. (2013, Mayo 7). CarScan USB Car Diagnostic Scanner for PC ELM327 OBD2 Instructions [Archivo de video]. Recuperado de https://www.youtube.com/watch?v=9Q9FRM9zeKY

Manuales:

ArduinoDev. (s.f.). OBD-II Adapter for Arduino. Recuperado de http://arduinodev.com/hardware/obd-kit/

Arduino Verkstad AB. (s.f.). Recuperado de http://castilla.verkstad.cc/es/course-literature/puerto-serie/

Software:

Arduino. (s.f.). Download the Arduino Software. Recuperado de http://arduino.cc/en/pmwiki.php?n=main/software

Download PuTTY. Recuperado de http://www.putty.org/

Librerías:

GitHub. (s.f.). ArduinoOBD libraries. Recuperado de https://github.com/stanleyhuangyc/ArduinoOBD/tree/master/libraries/OBD

Foros:

Arduino. [RET80] (2013, Agosto 17). Clearing the serial monitor [Foro]. Recuperado de http://forum.arduino.cc/index.php?topic=183020.0

Coches:

Fiat Dobló, (s.f.). En Wikipedia. Recuperado de http://es.wikipedia.org/wiki/Fiat_Dobl%C3%B2

Nissan Almera, (s.f.). En Wikipedia. Recuperado de http://es.wikipedia.org/wiki/Nissan_Almera

Seat Ibiza, (s.f.). En Wikipedia. Recuperado de http://es.wikipedia.org/wiki/SEAT_Ibiza

Definición de los parámetros:

www.equivalencias.info. (s.f.). Buscador equivalencias neumáticos. Recuperado de http://www.equivalencias.info/neumaticos/

autopista.es. (2009, Febrero 2). La velocidad real. Recuperado de http://www.autopista.es/noticias-motor/articulo/velocidad-real-46747.htm

Guioteca. (2010, Diciembre 6). La temperatura del motor, cómo funciona el sistema. Recuperado de http://www.guioteca.com/mecanica-automotriz/la-temperatura-del-motor-como-funciona-el-sistema/

Total Car Diagnostics. (2013, Febrero 1). Reading Performance Information Data (PID). Recuperado de http://www.totalcardiagnostics.com/support/Knowledgebase/Article/View/43/6/reading-performance-information-data-pid

MOTOR MAGAZINE. (2005, Marzo). Interpreting Generic Scan Data. Recuperado de http://www.motor.com/article.asp?article_ID=889

MAGNAFLOW. (s.f.). MAGNAFLOW CONVERTIDORES CATALÍTICOS OBDII. Recuperado de http://www.magnaflow.com/00espanol/02catalytic_converters/04basics/04convBasics.asp

OBD-CODES. (s.f.). What are fuel trims?. Recuperado de http://www.obd-codes.com/faq/fuel-trims.php

OBD-CODES. (s.f.). MAP (Manifold Absolute Pressure) Sensor. Recuperado de http://www.obd-codes.com/faq/map-sensor.php

e-auto. (s.f.). Sensor IAT-Sensor de Temperatura del Aire de Admisión. Recuperado de http://e-auto.com.mx/manual_detalle.php?manual_id=225

Ignition timing, (s.f.). En Wikipedia. Recuperado en http://en.wikipedia.org/wiki/Ignition_timing

MAXIMA.ORG. [SteVTEC] (2003, Julio 17). Ignition Timing Advance vs Knock Sensor Operation [Foro]. Recuperado de http://maxima.org/forums/4th-generation-maxima-1995-1999/170454-ignition-timing-advance-vs-knock-sensor-operation.html

mp3Car. [Vitaliy] (2010, Noviembre 30). OBD2 throttle position PID [Foro]. Recuperado de http://www.mp3car.com/engine-management-obd-ii-engine-diagnostics-etc/145273-obd2-throttle-position-pid.html

TORQUE WIKI. (2011, Diciembre 11). Throttle Position. Recuperado de https://torque-bhp.com/wiki/Throttle_Position

ITV:

Applus Iteuve. (s.f.). Periodicidad. Recuperado de http://www.applusiteuve.com/es/cada-cuanto-pasar-itv

Anexos:

Anexo 1:

ScanTool.net. (s.f.). Supported PIDs. Recuperado de http://www.scantool.net/obdwiz/

Anexo 2:

Arduino. (s.f.). Arduino UNO. Recuperado de http://arduino.cc/en/Main/ArduinoBoardUno

Anexo 4:

OBDCENTER. (s.f.). Recuperado de http://www.obdcenter.com/

CocheOBD2.com. (s.f.). Recuperado de http://www.cocheobd2.com/

INNOVA. (s.f.). Recuperado de http://www.innova.com/

DIAGNOSIS AUTO. (s.f.). Recuperado de http://www.diagnosisauto.com/

amazon. (s.f.). Recuperado de http://www.amazon.es/Diagn%C3%B3stico-Bluetooth-Interfaz-Detector-Tester/dp/B00CNZTTIO

Google Shopping. (s.f.). Recuperado de https://www.google.es/?gws_rd=ssl#q=OBD&tbm=shop

IMETRIK. (s.f.). Recuperado de http://www.imetrik.com/en/

MapmyIndia. (s.f.). Recuperado de http://www.mapmyindia.com/tracking

Sinocastel. (s.f.). Recuperado de http://www.sinocastel.com/

CARTRACK. (s.f.). Recuperado de http://www.cartrack.com/

Anexo 5:

-Programas informáticos para OBD:

HEALTECH. (s.f.). OBD-H01 – herramienta diagnóstica para motos Honda. Recuperado de http://www.healtech.es/obd-honda.html

HEALTECH. (s.f.). OBD-K01 – herramienta diagnóstica para motos Kawasaki. Recuperado de http://www.healtech.es/obd-kawasaki.html

HEALTECH. (s.f.). OBD-S01 – herramienta diagnóstica para motos Suzuki. Recuperado de http://www.healtech.es/obd-suzuki.html

AUTERRA. (s.f.). Dyno-Scan for Windows. Recuperado de http://www.auterraweb.com/dynoscan.html

-Aplicación móvil:

TORQUE WIKI. (s.f.). Main Page. Recuperado de http://torque-bhp.com/wiki/Main_Page

-Lista completa de averías:

OBD-CODES. (s.f.). OBD-II (Check Engine Light) Trouble Codes. Recuperado de http://www.obd-codes.com/trouble_codes/

-Sistema de codificación de averías:

MIAC. (s.f.). Códigos de avería OBDII – EOBD. Recuperado de http://www.codigoerror.com/